NASA TM-86327

NASA Technical Memorandum 86327

M68000 RNF Text Formatter User's Manual

Ralph W. Will and Carolyn Grantham

March 1985

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

==========

M68000

RNF

TEXT FORMATTER

USER'S MANUAL

==========

Adapted and Modified by:

Ralph W. Will
Carolyn Grantham

MARCH 1985

```
==================
|      R N F     |
|                |
| Text Formatter |
==================
```

Contents

(This page is intentionally blank.)

Introduction

About RNF

RNF is a program that formats document-oriented text such as this manual. It is designed to automate many of the tedious elements of typing, including breaking a document into pages with titles and page numbers, formatting chapter and section headings, keeping track of page numbers for use in a table of contents, "justifying" lines by inserting blanks to give an even right margin, and inserting figures and footnotes at appropriate places on the page. RNF greatly facilitates both preparing and modifying a document because it allows you to concentrate your efforts on the content of the document instead of its appearance and because it removes the necessity of retyping text that has not changed.

RNF is not restricted to formatting large and formal documents, however. Many people find it quick and convenient to produce short memos and other correspondence using RNF and an interactive editor.

It is important to understand that RNF is a formatter, not an editor. Both editors and formatters belong to the class of programs called "text processors". However, editors are used to create text files and to modify them by inserting new words, deleting old ones, swapping sentences, and so forth, while formatters like RNF modify text only by shoving words around to make them fit neatly on the page. RNF does not modify text by adding, deleting, or replacing lines or words. When you want to change the document, the modifications are actually made in the RNF input, using a keypunch or an interactive editor, and then RNF is executed again to reformat the document. Producing a finished document will ordinarily require repeating this process many times.

The original version of RNF was received from the Academic Computer Center of the University of Washington and runs under CDC NOS at NASA Langley. The version described here runs on the Renaissance Operating System (ROS) 2.0 on an M68000 facility in FDCD's Software Development Laboratory.This manual was adapted from the ACD manual, RNF - TEXT FORMATTER Users Manual(n-35) , February 1984.

About This Manual

This manual attempts to serve both as a primer for beginning users of RNF and as a reference manual for experienced RNF users. The presentation is organized into ten main sections:

1. Introduction

   Which you are reading now.

2. For the Beginning User

   A brief summary of RNF's features and most common commands, intended to introduce you to RNF's basic capabilities as gently as possible.

3. Basic Commands

   A narrative description of RNF's basic commands arranged roughly in order of complexity, intended to teach the intermediate user how to get the most utility from them.

4. Macros

   An explanation of RNF's macro capability, which lets you do more formatting with fewer commands.

5. Variables and Expressions

   The basics of using arithmetic variables and expressions to control formatting.

6. Specialized Commands

   How to use RNF to produce footnotes, figures,superscripts and subscripts, cross-references, hyphenation, lists, Roman numerals, and other assorted formatting effects.

7. List of Commands

   Concise descriptions of all RNF commands, in alphabetical order, intended as a reference guide for experienced users.

8. Examples

   Input and output showing the actions of most of RNF's commands and all of its major features.

9. Control Statements, Error Messages, and Utilities

   System control statements and utility programs needed to use RNF on ROS in the Software Development Laboratory at NASA Langley.

10. Appendices

    Additional details of certain specialized areas, and a quick-reference list of commands.

If you are just getting started with RNF, most of this manual won't make a bit of sense. So, just read the next section, "For the Beginning User" and look at the first few examples. Then read "How to Run RNF" in the section "Control Statements , Error Messages, and Utilities" and dive in and try it--there's nothing like a little experience.

If that brief introduction gives you enough information to comfortably handle your problem, fine. But if you're feeling cramped by the simple commands, then keep reading. The parts of "Basic Commands" from "How RNF Works" through "Environments" are mostly background, but they will reinforce what you have already learned and will fill in the gaps in the earlier presentation of the common commands. They may also give you some ideas on how to combine commands to get more interesting results.

When you are ready for some heavy thinking, read the parts on "Macros" and "Expressions". These provide the real power of RNF, but they are not easy to use. If you don't have much programming experience, you can expect to have some trouble understanding them. The best thing you can do is to imitate ours, try some, and see what happens--as we said earlier, there's nothing like a little experience.

This manual assumes that you know nothing about text formatting programs like RNF but that you are familiar with the ROS operating system and can use the interactive ROS text editor.

(This page is intentionally blank.)

For the Beginning User

RNF Features

RNF does the following things for you:

Pagination	RNF breaks text up into sequentially numbered pages. The format of a page is up to you--you can put the page number anywhere you want and you can have any number of heading lines at the top and bottom of each page.
Filling	With RNF, you don't have to worry about margins or line length while you are typing the document. Using margins that you specify, RNF runs together separate lines of input text to make lines of output that are more nearly equal in length. Filling can be turned off if you prefer to format some parts of the document yourself.
Justification	RNF will automatically justify filled lines by inserting blanks so that the right margin is even. This can be turned off if you prefer a "ragged right" margin.
Figures	If you have illustrations to paste in, RNF will leave blocks of space for them at attractive locations, with the body of the document flowing around them. For figures containing just text, such as tables of data, you don't even have to cut-and-paste. Just put the text of the figure in the RNF input and RNF will position it neatly on the page for you.
Footnotes	You can embed footnotes in the text and RNF will automatically move them to the bottom of the page.

RNF can also do hyphenation, via a "phantom hyphen" character, and will handle superscripts and subscripts on hardcopy terminals that can move the paper in half-line increments. It can cross-reference your document and produce a table of contents, and there are also built-in functions for numbering and titling chapters, sections, and lists.

Probably the most powerful feature of RNF is an easy-to-use macro facility that lets you combine built-in functions to create new commands or tailor old ones to match your application. The macro facility is particularly useful for redefining the page format, such as the number of title lines and the position and format of the page number. Macros are frequently used instead of the built-in chapter, section, and list features.

The most commonly desired features that RNF does not have are a) automatic hyphenation (not requiring phantom hyphen characters), b) automatic generation of an index, c) variable pitch justification (for those sophisticated terminals that can do that), and d) multi-column pages.

Most Commonly Used Commands

If you are just getting started with RNF, it may look rather forbidding. After all, how simple can a program be that requires a 106-page manual? The answer is that it is as simple as your application. While you are getting started with RNF, you won't be needing much of its power and you can get by with only a handful of commands. The ones you really need are the following:

Margin control: .LM, .RM, and .RIGHT

Spacing: .SP

Paragraphs: .P and .PP

Indented Lines: .I

"As is" Text and Significant Blanks: .ASIS and the pound sign (#)

Now is a good time to look first at page 55and then Example 1 on page 56, which illustrates the use of these commands in an RNF input file. This will also introduce you to the basic structure of an RNF input file. Tab stops, titling, underlining, and other capabilities are illustrated in the subsequent examples.

All of the commands illustrated in the examples are described in detail in the List of Commands starting on page 41.

Basic Commands

How RNF Works

You can think of RNF as a "programmable typewriter" that reads an input file, formats it, and writes an output file. The RNF input file contains all the text you want to appear in the final document and some RNF commands to control the formatting. As text is read from the input file, a word at a time, it is assembled into lines. Each time a new line is finished, RNF checks to be sure that there is space for it on the current page. If the page is full, RNF starts a new page by writing a title, subtitle, page number, or whatever. Then the newly finished line is written to the output file and the next word is read from the input file. The process continues until all the input has been read.

RNF commands that control the formatting of the lines and pages are interspersed with the text in the input file. Many of the commands simply set parameters that are used to control subsequent formatting, such as the left and right margins, page length, tab stops, and so on. Other commands are used to invoke special formatting functions at the point where the command appears. Examples of these are commands to start a new paragraph, to force a page eject, and to insert blank lines.

All of the previous functions are built-in, meaning that they are simply coded into the RNF program. You can also define commands of your own, called "macros", that allow you to invoke long sequences of the built-in functions with a single command. One special macro, named "FRCPAGE", is used internally by RNF to control what is done to end one page of the document and start a new one. By redefining this macro you can have complete control over the formatting of titles, subtitles, page numbers, etc.

Words

Probably the most important thing to keep in mind when using RNF is that it does not speak English--as far as RNF is concerned, a "word" is simply any sequence of nonblank characters. Thus, "word" is a word, as are "all-this-hyphenated-stuff" and "(&%$&)".

Blanks in the input file serve to separate words but usually do not have any significance of their own. RNF simply inserts one blank between words in the output, and then inserts more blanks as necessary if the lines are being justified.

On the assumption that it ends a sentence, a word with a period, question mark, or exclamation point after its last letter or digit will be followed by two blanks. For example, U.S.A. will ordinarily be followed by two blanks. (Extra blanks may be inserted by justification, of course.) You can eliminate the extra blank by flagging the character that causes it with an underscore. For example, U.S.A_. will be printed as U.S.A. followed by only one blank. If the problem occurs frequently (in a table of numbers, for example), you can turn off the extra blank with the .NOPERIOD command.

There are several ways for you to explicitly determine the placement of blanks in the document. The easiest way, for small amounts of text, is the "significant blank" shown in Example 2 (page 58). Besides occupying a fixed amount of space, significant blanks also prevent a word from breaking across two lines. For example, be###tween will print as be tween, with the "be" always on the same line as the "tween".

For larger amounts of text, such as a sample computer program, the easiest approach is to use "as is" text as shown in Example 1 (page 56). It lets you supply text that is already formatted, so that RNF simply copies it to the document. In rare instances, you may want to use the .SIG command (usually with .NOF selected also--see page 10). Both .ASIS and .SIG cause the output to be spaced like the input, but .ASIS causes subsequent commands to be ignored while with .SIG subsequent commands are still executed. For example, you can't do underlining inside "as is" text because underlining is controlled by a command (.U), but you can underline following .SIG .

In some cases, it is convenient or necessary to produce a single output word from two or more input words. One example would be to create [2], where the brackets are just text and the "2" is the result of an arithmetic computation. (See page 32 for the source of this example.) The command that suppresses the blank between two words (i. e., concatenates two words) is .X ("extend")--see Example 6 on page 68 or almost any macro in this manual. Most simple documents will not need .X .

Underlining and overstriking are described and shown in Example 6 (page 65). Notice that overstriking can be used to produce some interesting graphics like ■ (the characters X, O, H, and I overstruck).

A word of caution: the following characters have special meaning to RNF and should be used with care:

_ \ # / $. ? !

Special Characters and Flags, page 15, describe how to control the use of these characters.

Lines

The format of lines in the document is controlled by several parameters that can be set by RNF commands. These are the left and right margins, the filling and justification flags, and the tab stops.

The left and right margins are set by the .LM and .RM commands, respectively. The margins can be set either absolutely or relative to their current position. For example, the command .LM 5 sets the left margin to column 5, regardless of where it previously was, while .LM +5 moves it 5 columns to the right and .LM -5 moves it 5 columns to the left. .LM +0 does nothing and .LM 0 is illegal (there is no column 0). Simple usages of .LM and .RM are shown in Example 3 (page 58).

Also, you can use any "expression" (see Variables and Expressions, page 25) to set the margins relative to each other or to another parameter. For example, the command .RM $$LM+50 sets the right margin 50 columns away from the current left margin. The use of expressions is described fully in Variables and Expressions, page 25, but you should keep in mind throughout this manual that an expression can be used anywhere a command expects a numeric argument.

Centering (.C) and indentation (.I) are shown in Examples 1 and 2, respectively (pages 56 and 57). Note that the .C command does not have to be on the same input line as the text being centered--it just sets a flag that causes the next line to be centered before it is printed.

Tab stops are described fully in Example 4 on page 60.

Most documents have to be printed with a sizable blank space at the left of each page to allow for binding and punching. You can of

course provide this space with an appropriate .LM setting, but it is easier and better to use the special .RIGHT command instead. You can think of .RIGHT as simply picking up the entire document and sliding it to the right before printing it. This lets you start every document with the left margin at "column 1" (the default), and still have it printed with a blank left margin--see Example 1 for an illustration (page 56).

Justification is default; it can be turned off with .NOJ and turned back on with .J . Filling is also default; it is controlled with .NOF and .F .

Spacing

RNF input is always single spaced, but the output can be any spacing you want: single, double, triple, or wider. The spacing is easy to change and it is common to produce double- or triple-spaced draft copies and a single-spaced final version.

The overall spacing between output lines is set with the .SP command; ".SP 1" sets single spacing, ".SP 2" sets double, and so on. The spacing can be changed within a document, such as to have a single-spaced excerpt in a double-spaced thesis.

RNF is usually run in filling mode, in which input lines are run together to make output lines as long as possible without exceeding the right margin. This is handy for most text, but there are many situations in which you do not want input lines run together or would like to explicitly advance to the next output line. Common examples are when listing a mailing address and when putting titles on the columns of a table.

To turn off filling, use .NOF, shown in Example 4 (page 62). Following .NOF, RNF will advance to the next output line every time it finishes an input line. The last line of a paragraph is never filled, even if .F is on.

You can also explicitly advance to the next output line, which is called a "break" or a "carriage return" depending on how spacing is handled. The commands are .BR , which ignores spacing, and .CR , which uses it. For example, in a double spaced document (".SP 2"), the command .BR will print the current line and leave you positioned at the start of the very next line on the output page. A .CR will print the current line and then print a blank line to give double spacing. In a single spaced document, of course, there is no difference between .BR and .CR .

To save you the trouble of sprinkling .BR and .CR commands throughout your document, many commands do a break or a carriage

return before they do anything else. For example, a .PP command (see Example 3, page 60) starts a new paragraph, but it first does a .CR to end the current one. Similarly, .B (next paragraph) prints a blank line after first doing a .BR . The List of Commands starting on page 41 indicates which commands have implied .BR or .CR commands.

You can insert blank lines in the formatted document with the .B and .S commands. The difference between them is that .B ignores spacing whereas .S uses it. For example, in a double spaced document the command .B 3 will produce 3 blank lines while .S 3 will produce 6. If the spacing were set to single in a later version of the document (.SP 1), then both .B 3 and .S 3 would produce 3 blank lines. Both .B and .S are ignored at the top of a page because it is assumed that the page eject provides the visual separation intended by the blank lines. If you want blank lines at the top of a page, just precede the .S or .B command with a significant blank. (Significant blanks are shown in Example 2, page 58.)

If RNF is not autoparagraphing (see .AP command--Example 3 page 60), then a blank line is interpreted as a .B 1 command. This means that you can get blank lines in your output by simply putting blank lines in your input at the desired places.

Another way to insert blank lines is with the .VT command. This is a "vertical tab"--you specify the line number and .VT puts out enough blank lines to get you there.

Most documents will have sections of text that must be kept together on one page to protect the continuity or to avoid "widows". (A widow is a single line of a paragraph left hanging at the top or bottom of a page.) RNF provides several ways to do this. One simple command is .TP n ("test page"), where n is the number of lines you want kept together. This forces a page eject if there are less than n lines left on the current page, otherwise it does nothing. .TP does not break the current line--if a page eject is needed it is deferred until the current output line has been finished--so you can stick a .TP a few lines before the end of a paragraph if preventing a widow is important. (RNF automatically invokes .TP 2 at the beginning of each paragraph.) .TP is a very common command in macros, particularly for section headers.

Another way to keep text together is to bracket it with the .KEEP and .ENDKEEP commands. .KEEP is a little more restrictive than .TP in that .KEEP does break the current line, so it cannot be used in the middle of a paragraph. However, .KEEP has the advantage that it does not require you to count lines--RNF does the counting for you.

.TP, .KEEP, and .ENDKEEP are illustrated in Example 4, page 62.

Paragraphs

The two modes, manual and autoparagraphing (.PP and .AP commands), are described in Example 3 (page 60). Inside RNF, automatic paragraphing is implemented by simply generating a .PP command on your behalf whenever there is a blank in column 1, so the two modes are exactly the same except for the way a new paragraph is indicated. You may prefer either, depending on the type of document and your writing and editing style. Even in autoparagraphing mode, it may be convenient on occasion to use an explicit .PP to start a paragraph.

Example 3 also describes .P, the command that determines the format of the paragraphs. Note especially that the meaning of the indent parameter depends on whether it is signed. If the indent is unsigned, then it is interpreted as an absolute column number and the first line of each paragraph will start in that column regardless of where the left margin happens to be. If the indent is signed, then it is interpreted as being relative to the left margin at the time the paragraph starts. Relative indents are particularly handy to use if you change the left margin a lot, because they cause the paragraphs to look the same regardless of the margins.

For example: The command ".P +5 0 3" produces paragraphs indented fi ve spaces from the left margin, with no blank lines between paragraphs and at least the first three lines of each paragraph on a page. The command ".P +0 1 3" produces block paragraphs like the ones in this manual (no indention). The command ".P 15 1 3" would be very unusual, because it would cause the first line of each paragraph to start in column 15 regardless of the left margin, and ".P 0 1 3" would be illegal--there is no column 0.

Note that paragraphs with a "hanging indent" can be specified with a negative indent or with a column number to the left of the left margin. Example 3 shows only the negative indent form.

Pages

RNF automatically breaks your document into pages by counting lines, or you can force a new page at any point with the .PAGE command.

The size of an output page is controlled with the .PS command, which has two arguments: the number of lines on a page and the width of the page. The page width specified to .PS is really nothing more than another way of setting the right margin--it just saves you from having to put a .RM command at the start of your document.

In counting the number of lines per page, you should keep in mind that "line 1" is physically the fourth line on the page, because of the way that line printers execute a page eject. Thus, the maximum number of lines on standard 11-inch paper at 6 lines per inch is 63 (11*6-3=63). The default is 57 lines per page, which produces a 1-inch margin at the bottom. This manual uses .PS 57 68 .

When you are using a line printer for output, .PS is the only command you will ever need. However, if you are using a hardcopy interactive terminal, then you will also need to use the .LINES command. The problem that .LINES addresses is that most hardcopy terminals using continuous forms do not have a "page eject" function that RNF can use. Since normally some pages may be shorter than others (if .TP forces a page eject, for example), a hardcopy terminal will gradually lose alignment between the pages as produced by RNF and the paper running through the terminal. The cure for this is to have RNF make each page the same length by adding blank lines to the end. The .LINES command specifies how many physical lines each page should contain, including the top, middle, bottom, and blanks.

The argument to .LINES depends on the size of paper that the terminal is using and how many lines per inch it prints. For example, when using 11 inch paper on a terminal that prints at 6 lines per inch, the correct command is ".LINES 66" . For 8 1/2 inch paper at 8 lines per inch, it is ".LINES 68" , and for 8 1/2 inch paper at 6 lines per inch it is ".LINES 51".

Page Format

The standard page format provided by RNF is as follows:

Lines 1-3: Blank - This format actually provides 6 blank lines (a 1-inch margin) because "line 1" is physically the fourth line on the page.
Line 4: A title on the left and a page number in Arabic numerals on the right.
Line 5: A subtitle
Lines 6-7: Blank
Lines 8-n: Your text

The default title and subtitle are blank; they can be set to something else with the .TITLE and .ST commands, as shown in Example 5 (page 66). The page number is optional; it is present by default but can be turned off and on with .NONMP and .NMP . Pages are still counted even when the page number is not being printed. The page number can be set to a particular value with the .PNO command. It can be accessed through the predefined variable $$PAGE (see page 25) and through the .ASSIGNPN command (page 84).

This default page format is a reasonable compromise for many applications, particularly draft copies, but almost all polished documents demand something different. The page format is determined by a predefined macro named FRCPAGE, and you can get almost any format you can imagine by writing your own macro. Details on writing FRCPAGE macros start on page 21, but you'll have to read about macros in general first (see page 17).

Environments

A feature that contributes greatly to RNF's ease of use is the ability to save and restore the "environment". The environment is nothing more than the current settings for most of the flags and parameters that control formatting. Specifically, the environment contains the values of the following parameters:

> Left margin (.LM), right margin (.RM), paragraph parameters (.P), spacing (.SP), and tab stops (.TABS, .TAB).

It also contains the current settings of the following flags:

> Justification (.J), filling (.F), significance of blanks (.SIG), underlining (.U), periods in tabbed fields (.DOT), and hyphenation (.FLAGHYPH).

There are two common uses for saving and restoring the environment. The first is when you make a lot of changes to formatting parameters and then want to restore all the old values. In this case, the best commands to use are the .SAV and .RES commands. The second is when you want to restore a fixed environment several times to get a known starting point, such as at the beginning of each chapter of a document. Here, the best commands to use are .SAVPAG and .RESPAG or .SAVENV and .RESENV .

The environment-saving commands differ as follows: .SAV and .RES work with a pushdown stack of environments and can be nested up to 20 levels deep. Each .RES restores the environment that was saved by the matching .SAV . .SAVENV and .RESENV work with a table of up to 10 environments that are referenced by name. .SAVENV and .RESENV don't have to be used in pairs-- typically you .SAVENV a particular environment once and then .RESENV it many times. .SAVPAG and .RESPAG are like .SAVENV and .RESENV except that they reference a special "page environment". The page environment is just the environment in effect when the first page eject is done--it is provided for ease of writing FRCPAGE macros.

All the environment commands are very common in macros. See Example 3 (page 60) for an application of .SAV and .RES in normal text. A

typical use of the environment commands in a FRCPAGE macro is shown on page 21.

Special Characters and Flags

There are several characters that have special meaning to RNF. Each of these is under control of a flag that can be either on or off--if the corresponding flag is off, then the character has no special meaning and is just one more printable character. The characters and the commands to set their flags are:

Character	Name	Usage	Flag Command	Page
_	underscore	escape character	.FLAGESC	15
\	backslash	overstrike	.FLAGOVER	68
#	pound	significant blank	.FLAGSIG	58
/	slash	phantom hyphen	.FLAGHYPH	68
$	dollar	(leading) flags variables	(none)	
.	period	(leading) flags commands	(none)	
_		(trailing) ends sentence	.PERIOD	8
?	question	(trailing) ends sentence	.PERIOD	8
!	exclamation	(trailing) ends sentence	.PERIOD	8
_		(in column 1) ends "as is" mode	(none)	

The phrase "ends sentence" in the table means that the word containing the character will be followed by two blanks instead of one.

Except for phantom hyphen, all of these flags are on by default. Any flag can be turned off by prefixing the command with "NO", as in .NOFLAGSIG and .NOFLAGOVER .

In "as is" text (see Example 1, page 56), all of these characters are printable.

To use any of these characters as a printable character without having to clear its flag, simply precede it by the escape character underscore (_). For example, the RNF input:

    The pound character (_#) is a significant blank.

will be printed as:

    The pound character (#) is a significant blank.

For this to work, the escape character must be enabled with .FLAGESC (which is the default). Do not use the underscore indiscriminately, when it precedes a letter, digit, or colon a nonprintable control code results. Also, when one of the special characters follows a backslash for overstriking, an underscore is not needed. For example, the following RNF input line:

```
An "O" overstruck with a pound sign looks like:##O\#
```

will be printed as:

```
An "O" overstruck with a pound sign looks like: ⊕
```

Macros

Introduction

The macro facility is a very basic part of RNF. Macros are used to do the following:

- Define the page format by adjusting the number and contents of the header and footer lines, position of the page number, and so on.
- Extend the basic set of RNF commands (the built-in commands) by adding macros tailored to a specific application.
- Abbreviate commonly used series of RNF commands to reduce typing and make it easier to change the format of the document.
- Abbreviate commonly used phrases or long words.

The simplest way to think of macros is that a macro is a one-word abbreviation for a whole line of input. To those of you who are programmers, a macro is similar in concept to a subroutine or procedure. The power of macros comes from two additional features:

- The line of input represented by a macro can contain both commands and text.
- A macro can have "substitutable parameters" that let you change part of the input represented by the macro at the time the macro is called.

There are many situations in which a carefully considered macro definition will save you a great deal of typing, particularly for formatting tables, lists, and repetitious examples. Macros are also very useful for setting up headings for chapters, sections, subsections, and so forth. Some macros that have been found useful are shown in Examples 9 and 10 (pages 76 and 78). The FRCPAGE macro, which determines the page format, has a section of its own starting on page 21.

Simple Macros

There are two steps to using a macro: defining it and calling it. For example, a macro named "SDL" can be defined by the following command:

```
.MACRO SDL = Software Development Laboratory
```

The SDL macro can then be called by simply prefacing it with a period (.SDL), so that it is in the standard command format, and plugging it into your input. Whenever RNF encounters this new .SDL command, it will remove the .SDL and substitute "Software Development Laboratory". For example, the input line:

```
RNF: A Text Formatter for .SDL Computers
```

will be treated exactly as if it read:

```
RNF: A Text Formatter for Software Development Laboratory Computers
```

This example illustrates the simplest type of macro, one with no substitutable parameters. In general, you can define a macro like this with the command:

```
.MACRO name = stuff
```

where "name" is the name of the macro and "stuff" represents the actions to be performed by the macro. The entire macro definition must be contained on one RNF input line. The syntax requires the blanks to separate everything. The macro name is limited to 10 characters and must start with a letter or a period. The case of macro names is irrelevant--"xxx", "Xxx", and "XXX" represent the same macro. "Stuff" can contain almost anything, including text, commands, or calls to other macros, even macros that haven't been defined yet. The only restrictions are:

- You can't nest macro definitions. This means simply that a macro cannot contain another .MACRO command. It is legal and very common, however, to nest macro calls (i.e., to have one macro call another).

- All macros must be defined by the time they are actually called. You can refer to undefined macros when defining another one, but be sure they are all defined by the time the calling macro is invoked.

- Macros can't be called recursively (i.e., a macro can't call itself).

A macro definition is limited to one line of input. However, you can imitate a very long macro by simply breaking it up into smaller macros that are each short enough to fit on a single line, as follows:

```
.MACRO ONE = This is a lot of stuff that we would like to
.MACRO TWO = have in a single macro but can't because a
.MACRO THREE = macro is limited to one line and this is
.MACRO FOUR =  too much to fit.
.MACRO ALLOFIT = .ONE .TWO .THREE .FOUR
```

The macro call ".ALLOFIT" will then act as if ALLOFIT were defined to contain all four lines of text.

For almost all purposes (two exceptions will be described later), macro expansions are considered to take place on the same input line as the outermost macro call. This means that text following the macro call can serve as arguments to the final command in the macro. Thus, one simple but very handy use for macros is to "rename" a command to make it easier to remember. For example,

```
.MACRO RIGHTMARG = .RM
```

can be called with

```
.RIGHTMARG 65
```

Obviously, this approach is rather limited. Fortunately, macros with real substitutable parameters are only a little harder to define and use, and they are much more powerful.

Macros with Parameters

There are two ways to define macros with parameters:

```
.MACRO name n = text
   and
.MACRO name * = text
```

The "n" in the first form stands for the number of parameters used by the macro, which is a number from 1 to 7. In this case, the n words following the macro call are the parameter values. The "*" in the second form indicates that the macro has one parameter, which is the entire remainder of the input line on which the macro is called.

In the definition of macro "name" the formal parameters are called "name1", "name2", and so on, for however many parameters were declared. (In other words, parameter names are formed by appending a digit to the macro name.) These parameters are implicitly defined macros called "parameter macros"; the macro for which they are parameters is called their "owner". For example, consider the following macro definitions:

```
.MACRO ADDRESS 1 = NASA LaRC, MS .ADDRESS1 , Hampton, VA 23665
.MACRO XXX * = just as .XXX1 mud?
.MACRO SWITCH 2 = two things: .SWITCH2 and .SWITCH1
```

The first macro, ADDRESS, replaces its argument, .ADDRESS1, with the first word following ".ADDRESS" when ADDRESS is called. The second macro, XXX, accepts as an argument everything after ".XXX" in the call, and accesses it with ".XXX1" . For example, the following RNF input:

```
My work address is .ADDRESS 152D
Isn't this .XXX clear as
```

will generate the following output:

```
My work address is NASA LaRC, MS 152D , Hampton, VA 23665
Isn't this just as clear as mud?
```

The third macro defined above, SWITCH, illustrates the fact that there is no need to use the parameters in the macro definition in the same order that they appear in the call. (In fact, any parameter can be used more than once or even not at all.) SWITCH replaces its arguments, .SWITCH1 and .SWITCH2, with the two words following ".SWITCH" when SWITCH is called. The first of these words replaces every occurrence of ".SWITCH1" within the macro and the second replaces every occurrence of ".SWITCH2" . Notice how SWITCH uses its second parameter first and its first parameter second to perform its function of printing the values of its arguments in reverse order. For example, the following RNF input:

```
Consider .SWITCH that this
```

will generate the following output:

```
Consider two things: this and that
```

Some macros with parameters are shown in Examples 10 and 11 (pages 78 and 80).

Page Format -- The FRCPAGE Macro

The page format produced by RNF is entirely under control of a macro having the special name "FRCPAGE". It works like this: Each time RNF finishes an output line, it checks to see if there is room for the line on the current page. If there is, then RNF just prints the line and goes on to the next one. If there is not, then RNF moves the output line to another area for safekeeping and invokes the FRCPAGE macro, which looks something like the following:

```
.MACRO FRCPAGE = .BOT footer .TOP header .MID
```

The .BOT command causes RNF to prepare for a footer. This involves putting out any figures and footnotes that have been queued (see pages 31 through 33 for a discussion of figures and footnotes). Subsequent commands, represented by "footer", are then used to control the formatting of the bottom of the page.

The .TOP command signals the absolute end of text on the page. It causes RNF to issue enough blank lines and/or carriage control characters to physically get to the top of the next page. The commands after .TOP , represented by "header", control formatting the page header, if any.

The .MID signals the end of the header. .MID prints any figures that are pending (see Figures, page 31), then prints the output line that was saved for safekeeping at the beginning of all this activity. After .MID is done, RNF resumes scanning the input where it left off.

The default FRCPAGE macro used by RNF to set up the page format defined on page 13 actually looks like this:

```
.MACRO FRCPAGE = .BOT .TOP .HD .MID
.MACRO HD = .SAV .RESPAG .B 3 .TITLE1 ..PNO .BR .ST1 .B 2 .RES
.MACRO .PNO = $$PAGE=$$PAGE+1; .IF $$NMP .TAB $$RM .RT $$PAGE
```

The following is another fairly typical FRCPAGE macro:

```
.MACRO FRCPAGE = .BOT .FOOT .TOP .HEAD .MID
.MACRO HEAD = .SAV .RESPAG $$PAGE=$$PAGE+1; .B 3 .TOPPN .RES
.MACRO TOPPN = .SP 1 .C $$PAGE .B 3
.MACRO FOOT = .REM no page footer if page number at top
```

When combined with the commands ".PS 57 58", ".RIGHT 15", and ".SP 2", this FRCPAGE macro will produce output in the format required by the graduate school for text pages: blank borders of 1.5 inches on the left and 1 inch on all other sides, with the page number in the center at the top. The page number will be separated from the text by 1/2 inch (3 blank lines).

The .SAV, .RESPAG, and .RES commands inside macro HEAD deserve some comment. The .RESPAG resets the "page environment" (see page 14) so that .C centers the page number above the page as a whole, not just above the first line. This is necessary so that the placement of the page number does not depend on the left and right margins at the time the page eject happens. Without it, an indented paragraph that just happened to have a page eject in the middle would move the page number. The surrounding .SAV and .RES commands save and restore the environment in effect outside FRCPAGE. Without them, the effects of .RESPAG would propagate outside FRCPAGE.

In macro TOPPN, the purpose of setting single-spacing (.SP 1) is to prevent an extra blank line from being generated by the implied .CR that is part of .C .

There are some restrictions on changing the environment within FRCPAGE that will be discussed later. For now, let's resume the example. To get the page number at the bottom of the page in Roman numerals, the TOPPN and FOOT macros would be altered as follows:

```
.MACRO TOPPN = .REM nothing extra at top of page
.MACRO FOOT = .SAV .RESPAG .SP 1 .B 3 .C .FMT 4 $$PAGE .RES
```

The .FMT command is described in detail on page 40. Briefly, .FMT formats the value of its second argument according to a code given as its first argument. In this application, the value to put out is the page number and the 4 says do it in lower case Roman numerals.

Because the footer now requires four extra lines, the page length must be reduced to .PS 53 58 to get the correct margin below the page number. Because the header is four lines shorter, however, the amount of text material on a page is unchanged.

More complicated FRCPAGE macros can produce more interesting results, particularly if you use some expressions and conditional execution commands--see Variables and Expressions, page 25. For example, to get alternating page numbers--odd on the right, even on the left--you can use the following trick:

```
.MACRO FOOT = .SAV .RESPAG .B 2 .SETRT .EVENPN .ODDPN .RES
.MACRO SETRT = .TAB $$RM
.MACRO EVENPN = .IF ($$PAGE-($$PAGE/2)*2).EQ.0     $$PAGE
.MACRO ODDPN =  .IF ($$PAGE-($$PAGE/2)*2).NE.0 .RT $$PAGE
```

This looks much more complicated than it really is. The SETRT macro simply sets a tab at the right margin for possible later use by ODDPN. Macros EVENPN and ODDPN are then both executed, regardless of the page number, but only the appropriate one actually produces any output. Because RNF truncates the result of any division, the result of evaluating n-(n/2)*2 will be 0 if n is even and 1 if n is odd. So, for even page numbers, the .IF statement in macro EVENPN is satisfied and the page number goes out at the left margin.

Meanwhile, ODDPN produces no output because its .IF tests a false condition. For odd page numbers, the reverse is true, and ODDPN then puts out the page number via a right-justified tab to the right margin.

Conditional execution in the FRCPAGE macro can also make formatting control easier, by allowing you to define mnemonic commands. For example, consider the following macro definitions:

```
.NMP
.VAR $PNP .VAR $PNF
.MACRO FRCPAGE = .BOT .FOOT .TOP .HEAD .MID
.MACRO HEAD = .SAV .RESPAG $$PAGE=$$PAGE+1; .B 3 .TOPPN .RES
.MACRO TOPPN = .IF $PNF.EQ.0 .TOPPNA
.MACRO TOPPNA = $PNP=$PNP+1; .IF $$NMP .IF $PNP.GT.1 .SP 1 .C $$PAGE .B 3
.MACRO FOOT = .SAV .RESPAG .FOOTA .RES
.MACRO FOOTA = .IF $PNF.EQ.1 .SP 1 .FOOTB
.MACRO FOOTB = $PNP=$PNP+1; .IF $$NMP .IF $PNP.GT.1 .B 3 .C .FMT 4 $$PAGE
```

In these macros, the variables $PNP and $PNF are used to control the presence and location of the page number. $PNP controls the presence of the page number--if $PNP is set to 0, the page number will be skipped on the next page and printed thereafter. $PNF controls the location of the page number--if $PNF is 0, the page number will be printed in Arabic numerals at the top of the page, while if $PNF is 1, the page number will be printed in Roman numerals at the bottom of the page. These formats correspond to the graduate school requirements for text pages and preliminary pages.

Of course, it is a little hard to remember that "$PNP=0;" prevents the page number from being printed and "$PNF=1;" causes Roman numerals at the bottom of the page. So, we define three more macros simply for their mnemonic value:

```
.MACRO PRELIMF = $PNF=1; .PS 53 58
.MACRO TEXTF   = $PNF=0; .PS 57 58
.MACRO SKIPPN = $PNP=0;
```

The result of this string of macro definitions is to define a set of three extremely convenient commands specialized for thesis formatting:

.PRELIMF causes subsequent page numbers to be printed in Roman numerals at the bottom of the page.

.TEXTF causes subsequent page numbers to be printed in Arabic numerals at the top of the page.

.SKIPPN causes the page number to be skipped on the next page. The page is still counted, and printing of the page number will be resumed automatically on subsequent pages.

Now that you have seen how to write correct FRCPAGE macros, we must talk about the restrictions so that you can avoid writing incorrect ones. First, note that you do not need to make any special effort to avoid extra page headers and footers at the ends of your document, because RNF treats the first and last invocations of FRCPAGE specially. On the first page of the document, FRCPAGE acts as if it were coded:

```
.MACRO FRCPAGE = .TOP header .MID
```

Similarly, the last invocation of FRCPAGE, to do the final page footer, acts as if it were coded:

```
.MACRO FRCPAGE = .BOT footer
```

Thus, you do not have to write .IF statements to explicitly avoid spurious stuff at the beginning and end of your document-- RNF handles this for you. However, this special handling does introduce some restrictions on the use of environment-changing commands within FRCPAGE. Basically, you cannot make a change in the environment between .BOT and .TOP (the footer) that is supposed to be undone between .TOP and .MID (the header). In particular, the following would not work correctly:

```
.MACRO FRCPAGE = .BOT .SAV footer .TOP header .RES .MID
```

The problem with this definition is that, on the first page, the .SAV would be skipped because it is between .BOT and .TOP . As a result, the .RES would be in error because there would be no .SAV to match it.

There are some other restrictions regarding FRCPAGE. The main one is that the .BOT, .TOP, and .MID commands should not be used outside FRCPAGE--it is legal but hard to handle. You should be careful to make .BOT the first command in the macro and .MID the last command. Certain commands will work correctly before the .BOT and commands are sometimes legal after the .MID, but both situations are governed by a plethora of complicated conditions. In general, none of these usages are checked for validity--mistakes will produce bad output instead of coherent error messages.

You can call FRCPAGE explicitly yourself, but it is usually better to use either the the .PAGE or .ENDPAGE commands. .PAGE is the command usually used--it simulates the sequence .CR .FRCPAGE, which ends the current page and starts a new one. In a few very complicated situations where both footers and headers change between chapters, you may need .ENDPAGE . Its action is to end the current page but not start a new one. Typically you would use .ENDPAGE to force a footer, then redefine FRCPAGE to get a new format for the start of the new chapter.

Variables and Expressions

Simple Variables and Arrays

The ability to do arithmetic on numeric variables is another RNF feature that contributes greatly to its flexibility. Numeric variables are very handy (and are used internally) for keeping track of page numbers, chapter numbers, header levels, and so forth, and for adjusting formatting parameters such as the margins and tab stops. For example, the variable $$LM is used internally to hold the left margin, so the command ".TAB $$LM+10" can be used to set a tab stop 10 columns from wherever the left margin happens to be.

Variable names begin with a dollar sign ($), followed by up to 10 letters, digits, or dollar signs. The case of variable names is not significant--$abc and $ABC are the same variable. The first character after the leading $ cannot be a digit. This restriction prevents $12 from being interpreted as a variable, which would be a nuisance.

There are a number of predefined variables used by RNF for various bookkeeping functions. Notice that all the predefined variables start with two dollar signs. To avoid problems, you should not define any variables of your own that begin with two dollar signs. The predefined variables, their meanings, and the commands that set them are:

$$PAGE	page number	(.PNO)
$$OLNO	output line number	(no command)
$$ILNO	input line number	(no command)
$$LM	left margin	(.LM)
$$RM	right margin	(.RM, .PS)
$$SP	spacing	(.SP)
$$NMP	page numbering flag	(.NMP)
$$CH	chapter number	(.CH)
$$ATITLE	automatic titling flag for chaptering	(.ATITLE)
$$HL	header level, array of 6 elements	(.HL)
$$LIST	list level, array of 6 elements	(.LIST, .LE)

These variables can be examined or changed at any time by using them in expressions. Of course, changing these values will affect the

operation of the built-in RNF functions that use them--you generally should use the standard commands rather than changing the values directly.

To declare variables of your own, use the .VAR command, which has the format:

```
.VAR $name
   or
.VAR $name = value
```

Note that the variable name is both declared and used with a leading dollar sign ($). This is different from macro names, to which you must add a period when you call them. The function of .VAR is to allocate space for the variable, give it the specified name, and initialize its value. The first form initializes the value to zero; the second form uses the specified value. (More will be said about this second form later--it has special uses.)

You can declare only 200 variables in addition to the predefined variables. If a variable is "declared" more than once (i.e., appears in more than one .VAR command), it still uses only one space.

Variables declared with .VAR can hold only a single value. Arrays can also be declared, by using the .ARRAY command as follows:

```
.ARRAY $name n
```

This command declares an n+1 element array named $name. The array elements are indexed from 0 through n and are referenced as $name[0] through $name[n] . The subscript can be any arithmetic expression, and an array element can be used anywhere that a simple variable can. If the array subscript is omitted, then the reference is to the zeroeth element--$name and $name[0] are exactly the same. It is very common to take advantage of this fact by using the zeroeth element of an array as an index into the other elements. For example, if $name[0] is set equal to 5, then $name[$name] references $name[5] . This merely avoids your having to define a separate variable to use as an index--there is no advantage in space or execution time.

Array space is allocated in the same tables that hold simple variables, so the n+1 elements of an array count against your total space of 200 variables.

Expressions

Almost all nontrivial uses for variables and arrays involve expressions that compute numeric values by combining variables, numbers, and operators. For the most part, RNF expressions look like any other arithmetic expressions. The biggest difference is that RNF expressions cannot have any embedded blanks. For example, the following are legal expressions:

```
$$RM
($SIZE-15-$$OLNO)/2
($$PAGE/2)*2
```

The following is not legal because of the embedded blanks:

```
$$RM - 15
```

It is also a good idea to use parentheses freely to eliminate any ambiguity in expressions involving both multiplication and division. This is required because RNF evaluates multiplication and division from right to left--the reverse of the way it is usually done. For example, the expression $$PAGE/2*2 will be evaluated as $$PAGE/(2*2) . To be interpreted as you would ordinarily expect, it must be written as _($$PAGE/2)*2 .

Incidentally, all RNF calculations are done in integer mode--division truncates the result.

Expressions can be used either as arguments for RNF commands or by themselves. If an expression appears by itself (i.e., is not a command argument), then its value is printed. For example, the sequence:

```
.LM 15 .RM $$LM+50
The current right margin is $$RM .X .
```

will print out "The current right margin is 65." (Recall that the .X command concatenates two words.)

Expressions occurring by themselves are evaluated only if their first character is a dollar sign. For example, the expression $A+2 occurring by itself will be evaluated before it is printed; the expression 2+$A will not be. If $A has the value 3, then $A+2 will produce "5", while 2+$A will produce "2+$A". In the rare case you need to force an expression to be evaluated, you can use the syntax $(expr) or use the .FMT command (see page 40).

An RNF expression can also be used to assign a new value to a variable. For example, the expression:

```
$$PAGE=$$PAGE+1
```

will increment the page number. (Remember that $$PAGE is the predefined variable used to hold the page number.) An assignment like this will also print the new value as a side effect. To keep the value from being printed, add a semicolon after the expression, as in:

```
$$PAGE=$$PAGE+1;
```

As usual, embedded blanks are not permitted; the semicolon must immediately follow the expression.

To use the value of a macro argument in an expression, you have to be a little indirect. The trouble is that RNF's expression evaluator does not know how to recognize and expand a macro call. For example, suppose we want to change the value of the predefined variable $$PAGE using a macro. The obvious solution is to define the macro as follows:

```
.MACRO PAGENO 1 = $$PAGE=.PAGENO1;
.PAGENO 15
```

However, this will not work - a call to this macro will simply yield an error when the expression evaluator encounters the period.

Accessing a macro argument in an expression generally requires two steps: assign the value to an intermediate variable, then use that variable in the expression. For example, the following will work:

```
.MACRO PAGENO 1 = .VAR $JUNK = .PAGENO1 $$PAGE=$JUNK;
.PAGENO 15
```

Another example of using the value of a macro argument in an expression can be found in Example 10 on page 78.

Conditional Execution

Expressions can contain logical operations in addition to arithmetic operations. These logical operations are a little unusual in that they check logical conditions but return numeric results. A true

condition is represented by a value of 1, false by a value of 0. For example, the expressions:

```
4.EQ.5
(4.EQ.4)+(5.EQ.5)
((4.EQ.4)+(5.EQ.5))*(6.EQ.99)
```

have values 0, 2, and 0 respectively. The command that checks "true" and "false" actually checks for nonzero and zero. An arithmetic multiply (*) is equivalent to a logical "and" while an arithmetic add (+) is equivalent to a logical "or". For logical "not", there is a special operator, the pound sign (#).

To use the results of these logical operators, there is an .IF command that controls conditional execution. The syntax for .IF is:

```
.IF expr anything
```

where "expr" represents any expression and "anything" is literally any legal RNF input. The behavior of .IF is quite simple--if the value of the expression is zero, then the entire remainder of the input line or macro call containing the .IF is skipped. If the value of the expression is nonzero, then the rest of the line is executed as usual. For example:

```
.IF 1.EQ.1 Print this .IF 2.EQ.1 and this .IF 1.EQ.1 stuff
```

will produce just "Print this". The first condition is true, so execution continues, putting out "Print this" and encountering the second .IF . The second condition is false, so the entire remainder of the line is skipped and the third .IF is never even checked.

For the purpose of conditional execution, macro expansions are not considered to be on the same line as the outermost macro. This is one of the exceptions mentioned in the description of macros. A .IF testing a false condition will terminate only one level of macro call. For example, the sequence:

```
.MACRO INNER = Here .IF 1.NE.1 and there
.MACRO OUTER = .INNER and everywhere
.OUTER else
```

will produce "Here and everywhere else"--the false .IF terminates only the innermost macro. Because of this convention, the expression being tested by the .IF cannot be contained in another macro; the expression must immediately follow the .IF on the same line.

Conditional execution is usually used in macros to make them more general. The standard macros, for example, use .IF to test several flags that are set by other commands. An interesting use of .IF is shown on page 22, in connection with the FRCPAGE macro and placement of the page number. Other macros that make use of conditional execution can be found in Example 11 on page 80.

Looping

To go along with its arithmetic variables and expressions and its conditional execution, RNF has a looping command. The command is ".AGAIN"--its action is to cause RNF to start scanning again at the beginning of the current input line or macro. For example, the sequence:

```
.VAR $I
.IF $I.LT.10 $I=$I+1 .AGAIN
```

will print out "1 2 3 4 5 6 7 8 9 10". (Remember that ".VAR $I" initializes $I to zero and that the value of the expression "$I=$I+1" is printed each time it is evaluated because it is not terminated with a semicolon.) To nest loops, you must use macros, as in:

```
.VAR $I .VAR $J
.MACRO DOIT = $I .X . .X $J=$J+1 .IF $J.LT.2 .AGAIN
$J=0; $I=$I+1; .DOIT .IF $I.LT.3 .AGAIN
```

The output of this will be "1.1 1.2 2.1 2.2 3.1 3.2" . The .AGAIN command is mostly useful in macros dealing with sequentially numbered sections. For example, the built-in .HL command, which is described in Example 7 (page 70), can be mimicked by the following string of macros:

```
.MACRO HL 1 = .VAR $HLN = .HL1 .VAR $HLI .HLA .HLB .HLC .HLD .HLE .HLF
.MACRO HLA = .TP 8 .RESPAG .B 3 .IF $HLN.GT.$$HL $$HL[$HLN]=0;
.MACRO HLB = $$HL=$HLN; $$HL[$$HL]=$$HL[$$HL]+1;
.MACRO HLC = .IF $$CH.GT.0 $HLI=0; $$CH
.MACRO HLD = .IF $$CH.EQ.0 $HLI=1; $$HL[1]
.MACRO HLE = $HLI=$HLI+1; .IF $HLI.LE.$$HL .X . .X $$HL[$HLI] .AGAIN
.MACRO HLF = .X ## .X
```

This string of macros will execute significantly more slowly than the built-in HL command, but the time spent processing section headers is almost never significant anyway. If you don't like the standard .HL command, you can probably get something you do like by modifying the above macros and using them instead.

Specialized Commands

Figures

The figure facility provided by RNF comes very close to the ideal that "a formatter should act like a smart typist". It virtually eliminates any need to cut-and-paste figures containing just text, such as those shown in Example 12 (page 82). Consider how these would have been described to a human typist. You probably would say something like:

> "Ok, now here is the text for these two figures. Please type them up neatly and set them aside, but remember how many lines they contain. Then, while you are typing the section that refers to them, look for a place to put the figures where they will be tidy, such as at the top or bottom of a page. When you get to a good place, then copy the figures onto the page just the way you originally typed them."

Instructions like these might not get you output that you would consider perfect, but it would certainly be acceptable. If you wanted to make the document prettier, then you could come back and give more explicit instructions on where to put the figures in the final copy, after all the content revisions were in and the pagination was set.

RNF works the same way. There are two commands, .DEFFIG and .ENDFIG, that are used to bracket the text of a figure to "define" it and give it a name. Between .DEFFIG and .ENDFIG, you can put any sort of formatting commands and text you want--RNF formats output lines just as usual, except that instead of writing them directly to the output file it puts them on a scratch file instead. When the figure is finished (when .ENDFIG is encountered), RNF makes a note to itself of how many lines the figure contained, then goes back to whatever it was doing when the .DEFFIG command was encountered.

The output file will show no evidence at this point that anything has happened--the only effect is that RNF has an extra table entry and some text salted away on a scratch file. The visible effects start when the figure is "called" by a .FIG command. At that time, RNF calculates a good place to put the figure, based on the figure's size, the current line number, and whether any other figures have been called but not printed. RNF then makes a note to itself that

"queues" the figure to be printed and goes on with whatever it was doing. When the selected place is reached, RNF copies the figure to the output (it is already formatted, remember), and then resumes the document text that surrounds the figure.

Now go read Example 12 on page 82. Most of the details of using figures were moved there to make the example long enough to span two pages.

There are several commands that cannot be used inside a figure. The main ones are .DEFFIG, .KEEP, .FOOTNOTE, and .TP . .PAGE is allowed inside a figure--it has no immediate effect, but causes a page eject after the figure is finally printed.

Footnotes

Footnotes are handled by RNF as a special case of the general figure facility. The basics of using footnotes are both explained and illustrated in Example 13, page 86, so you should go read that immediately.

There is one important restriction on footnotes: they cannot be used inside a figure (between .DEFFIG and .ENDFIG), inside another footnote (between .FOOTNOTE and .ENDNOTE), or inside a keep block (between .KEEP and .ENDKEEP).

If your document has many footnotes, you may wish to define a macro to take care of numbering them consecutively, so that you can insert more or delete some painlessly. The following macro would be typical:

```
.VAR $FN
.MACRO FN = .FNA .FNB
.MACRO FNA = $FN=$FN+1; .X .BRACKFN .FOOTNOTE .SAV
.MACRO FNB = .RESPAG .LM +4 .I -4 .BRACKFN .X
.MACRO BRACKFN = [ .X $FN .X ]#
.MACRO ENDFN = .B 1 .RES .ENDNOTE
```

With these definitions and margins of 6 and 50 (including the page environment), the following sequence:

```
This sentence is referenced. .FN This is the stuff
that references the sentence above.  Notice how it is
done with a hanging indent. .ENDFN This follows the
referenced sentence. .FN And this is a second footnote.
.ENDFN
```

produces the following output:

> This sentence is referenced.[1] This follows the referenced sentence.[2]
>
> [1] This is the stuff that references the sentence above. Notice how it is done with a hanging indent.
>
> [2] And this is a second footnote.

Ordinarily, the footnotes are separated from the nearest main text above them by two blank lines (independent of spacing). If this is not suitable for your application, you can change it by defining a figure named "FOOTBREAK". For example, invoking the following macro:

```
.DEFFIG FOOTBREAK .B 2 ---------- .B 1 .ENDFIG
```

will cause the footnotes to be separated from the main text by two blank lines, a row of dashes, and another blank line.

Control Sequences (Subscripts and Superscripts)

There are many types of devices that can produce output containing superscripts and subscripts. Most of these are printing terminals of the "daisy wheel" type, which communicate with the computer over phone lines at a rate of 30 characters per second. (The name "daisy wheel" comes from the shape of the print mechanism.) However, although all daisy wheel terminals can print superscripts and subscripts, the way in which a program controls such printing is different for virtually all of them. Most programs therefore support only one or a few brands of terminals.

RNF has no restrictions on what brand of terminal you can use. Superscripts and subscripts are implemented in RNF via a very general "control sequence" facility, which can also be used to control other special hardware features such as page ejects.

First, some background is required. All terminals communicate with the computer using a set of 128 different codes called the "ASCII" character set (American Standard Code for Information Interchange). Of the 128 codes, 95 are used to represent the "printable" characters--letters, digits, punctuation, and special characters. The remaining 33 codes are not usually printable, but are used to control special hardware features, such as moving the paper 1/2 line to do a superscript. RNF output usually contains only the 95 printable characters, but you can force RNF to put out almost any of

the other codes (ASCII DELete, code 7F hexidecimal, cannot be obtained). Thus, you can tailor RNF output to correctly control any type of terminal by choosing the control sequences needed by that terminal.

The way you make RNF put out an ASCII control code is by using a two-character combination composed of an underscore and another character. For example, the combination "_0" (underscore zero) will cause RNF to output an ASCII "ESCape" character-- code 1B hexidecimal. The character following the underscore determines which ASCII code is produced, as follows:

RNF Input	ASCII Output	RNF Input	ASCII Output
_:	NUL, 00 HEX	_0	ESC, 1B HEX
_A	SOH, 01 HEX	_1	FS, 1C HEX
_B	STX, 02 HEX	...	
...		_4	US, 1F HEX
_Y	EM, 19 HEX		
_Z	SUB, 1A HEX		

There are some restrictions. The RNF escape character must be enabled with .FLAGESC . This is the default and you needn't worry unless you have turned it off with .NOFLAGESC . Control sequences can't be used in "as is" text (same problem as underlining - see page 8), and they can't be used in display code output.

There is more to consider, however. Obviously, there is no way for RNF to infer that the "8" in ESCape-8 is not just another printable character. RNF will therefore count the "8" (and the ESCape too, for that matter) as being just another character on the output line. Unfortunately, this means that the "8" will certainly affect justification and may even force a new line. The resulting output would be a little strange.

To solve this dilemma, there is a pair of special RNF commands, .CTLSEQ and .NOCTLSEQ, that are used to bracket sections of invisible "text" such as control sequences. When RNF encounters .CTLSEQ, it stops checking margins and counting characters for justificaton. Checking and counting is resumed by .NOCTLSEQ . To avoid having RNF insert extra blanks to separate "words" (even invisible ones), control sequences are usually concatenated with the text around them via the .X command.

The following macro definitions illustrate the use of .CTLSEQ, .NOCTLSEQ and .X. Macros similar to these may be used to produce additional character sets, printer control characters and to change character fonts in an RNF output file that is to be printed on the Data South 220 letter quality printer in the Software Development Lab at LaRC.

```
.MACRO FEED = .CTLSEQ _J .NOCTLSEQ .X
.MACRO FONT16 = .CTLSEQ _0$16M .NOCTLSEQ .X
```

```
.MACRO FONT10 =  .CTLSEQ _0$1M .NOCTLSEQ .X
.MACRO TOGREEK = .CTLSEQ _0(G .NOCTLSEQ .X
.MACRO TOASCII = .CTLSEQ _0(B .NOCTLSEQ .X
```

For example, the following RNF input

```
.CR .LM +15 .FONT16
Here is an example that selects a new character font for printer
output, executes a line feed .FEED , and selects a new character
set (greek !). .CR  .TOGREEK this line is GREEK - abcdefghijk... .
.CR .TOASCII
Don't forget to reset the character set(ASCII) and the font size
before continuing the program.  .CR  .FONT10
.LM -15
```

will produce the following output

> Here is an example that selects a new character font for printer output, executes a line feed , and selects a new character set (greek !). ΤΘΙΣ ΛΙΝΕ ΙΣ GREEK - ΑΒΓΔΕΖΗΘΙjK... . Don't forget to reset the character set(ASCII) and the font size before continuing the program.

Please note that character fonts should be used with care. Also note that font changes within a document remove the capability to print the document in an arbitrary font and size, since it now must be known at RNF time which font and size to reset to.

Cross-References

If you work with documents that have to be modified frequently, one of the most endearing features of RNF is its ability to maintain accurate cross-referencing and tables of contents no matter how the pagination changes. In theory, the idea is that you always refer to other points in the document by name, not by page number, and leave it to RNF to associate the correct page number with the name. In practice, there are two ways to handle the association: using variables and using the .DEF and .REF commands.

Using variables is the more primitive of the two and is probably easier to understand. It is also cheaper to use, but unfortunately it is restricted to backward references, in which a named point is defined before it is referenced. To define a named point using a

variable, just pick a variable name and insert the following commands at the point to which you want to refer:

```
.VAR $var .ASSIGNPN $var
```

Then, later in the document, you can use the variable to access the referenced page number, as in:

```
... as previously described on page $var .X .
```

If the definition was on page 7, then $var will have the value 7, and the output produced will be:

```
... as previously described on page 7.
```

The second method, illustrated in Example 14 (page 88), is to use the special .REF and .DEF commands. (They are actually macros, but that will be described a little later.) To define a named point using .DEF, just pick a name (not a variable name--no dollar sign) and insert the following command at the point to which you want to refer:

```
.DEF name $$PAGE
```

To reference that definition, you can insert at any point in the document the command:

```
.REF name placeholder
```

The "placeholder" is any string that contains exactly as many characters as the defined value will have. For example, if name will have a defined value of "26", then placeholder can be anything with two characters, such as "xx". The function of the placeholder is to enable RNF to correctly justify the line containing the reference. In some cases the length of placeholder will be in doubt, such as when you are not sure whether the defined point will be on page 9 or page 10. In those cases, the easiest approach is to pick one and try it--the situation doesn't come up very often and it only takes one run to fix it if you miss.

There are several points to remember about .DEF/.REF.

1. There is a limit of 500 defined names with a total length of 7000 characters for all names and values combined. There is no limit on the number of references.

2. Because of a limitation in RNF's design, you cannot use .RT and .CT to right-tab or center-tab a reference. The .T command can be used. (This is not a severe restriction, since you have to know the length of the reference anyway.)

3. When .REF is used, the next output will immediately follow the referenced value. For example:

```
.DEF JUNK 15
What is the value of .REF JUNK xx ?
```

will produce:

```
What is the value of 15?
```

Notice that the question mark is adjacent to the "5". In most uses, the reference will be followed by punctuation, so this action is convenient. In cases where you would prefer a space left, just follow the .REF command with a .NOX, as in:

```
The value of .REF JUNK xx .NOX is meaningless.
```

To understand the need for placeholder and the REF parameter, you have to know how .REF and .DEF work. .REF and .DEF are actually macros -- for ASCII output files their definitions are:

```
.MACRO E! = .ESC
.MACRO E? = .CTLSEQ
.MACRO N? = .NOCTLSEQ
.MACRO D% = % .X .FMT 1 4
.MACRO R% = % .X .FMT 1 18

.MACRO DEF 2 = .E? .E! .X .D% .X .DEF1 .X .E! .X .DEF2
(continued)          .X .E! .N? .X
.MACRO REF 2 = .E? .E! .X .R% .X .REF1 .X .E! .N? .X .REF2
(continued)          .X .E? .E! .N? .X
```

In general, these macros are called by:

```
.DEF name value
and .REF name placeholder
```

The immediate result is that .DEF and .REF propagate the following sequences into RNF's output file:

```
<esc> %D name <esc> value <esc>
and <esc> %R name <esc> placeholder <esc>
```

These sequences are for an ASCII output file; the <esc> represents an ASCII ESCape.

For .DEF, the entire output is a RNF "control sequence" because it is surrounded by .CTLSEQ and .NOCTLSEQ commands. RNF therefore considers it to be invisible and calculates margins and justification as if the definition were not there. The same is true for .REF, except that the placeholder is excluded from the control

sequence and RNF therefore calculates margins and justification to allow for the number of characters in placeholder. RNF cannot use the number of characters in the definition because it has no idea at the time of the reference how many characters that might be. (In fact, RNF has no idea it is even doing a reference--.REF is just another macro as far as RNF is concerned.)

With these macro definitions, there is absolutely no restriction on what "value" can be as long as it can be represented in the .DEF call by a single word. (This restriction is imposed by the rules for macro parameter substitution-- see page 19.) Of course, you can always represent anything by one word by writing another macro. For example, to make "value" be the page number expressed in Roman numerals, you could use:

```
.MACRO ROMANPN = .FMT 4 $$PAGE
.REF PAGEREF .ROMANPN
```

Hyphenation

RNF handles hyphenation in a rather crude manner that is just barely good enough to live with. RNF knows when to hyphenate (at the end of a line) but does not know where to hyphenate (between syllables). So, you have to explicitly mark every place that you are willing to allow hyphenation.

This is done by turning on hyphenation with the .FLAGHYPH command and marking each place you will accept hyphenation with a "phantom hyphen" character, the slash (/). When RNF generates its output, the phantom will be removed if it occurs in the middle of a line and will turn into a hyphen if it occurs at the end of a line. Inserting phantom hyphens is clearly no easier than simply breaking the words yourself and inserting real hyphens, but it does protect you from the chance that subsequent document changes will move a word from the end of a line to the middle.

For example, the following RNF input word:

```
hy/phen/ated
```

would be printed either with or without hyphens, depending on where it happened to fall on the output line:

```
Left margin        Middle          Right margin
   +-----------------------------------+

                  hyphenated       hyphen-
ated                                  hy-
phenated
```

RNF will not automatically break a word that is already hyphenated. If you want to allow "built-in" to break at the hyphen, it must be coded as "built-/in". The combination "-/" is special-cased to produce only one hyphen even if the word breaks across two lines.

In "as is" text or if hyphenation is turned off with .NOFLAGHYPH, the slash is just another printable character. Note that .NOFLAGHYPH is the default--you must explicitly allow hyphenation by including a .FLAGHYPH command.

You should not bother marking hyphenation when you are working with rough drafts, because there will be so many places to mark. Wait until all the content changes have been made so that the layout is fixed, then look at the document and pick out particularly bad places. There will typically be only one or two per paragraph. Fix those by putting in phantom hyphens and try again. A few more bad places will appear as the hyphenation allows words to shift around, so you may have to make two or three passes to make the document look right.

Miscellaneous Commands

The following commands, although important and convenient, are best illustrated by example and thus don't need much introduction:

If your document has a hierarchical organization with sections inside sections, you may like the .HL command for formatting titles at the various header levels. See Example 7 on page 70.

If you have many numbered lists, you may like .LIST and its cohorts .LE and .ELIST--see Example 8 on page 74.

If you don't want to write your own macros to control formatting of chapter titles, then try the .CH command--see Example 7 on page 70.

If your document is frequently updated and its readers must be alerted to changes in its contents, then change bars may be useful--see Example 15 on page 90.

If you want to put comments in your RNF input to document some subtle RNF commands or to identify sections for your own use, then you need .REM --its syntax is:

```
anything .REM stuff to be ignored
```

.REM just causes RNF to skip the remainder of that input line. Any text or commands before the .REM are processed as usual.

If you need Roman numerals, say for page numbers in an introduction, then you need the .FMT command. It can also be used to number lists or sections alphabetically instead of numerically. The syntax is:

```
.FMT n1 n2
```

.FMT formats the value of n2 according to the format code specified by the value of n1. Both n1 and n2 may be constants, variables, expressions, or macros that expand as expressions. Legal values of n1 are 0 through 4, with the following meanings:

n1	Format n2 as
0	Arabic numerals (.FMT 0 n2 is the same as just n2 except that it forces n2 to be evaluated.)
1	upper case alphabetic characters, selected from the CDC character set by using n2 modulo 64.
2	lower case alphabetic characters
3	upper case Roman numerals (I, V, and X only)
4	lower case Roman numerals (i, v, and x only)

.FMT is illustrated in connection with the FRCPAGE macro on page 22 and is used in Example 10 on page 78.

List of Commands and Command Format

Any sequence of non-blank characters beginning with a period (.) is taken by RNF to be a command. If the command is not recognized, an error message is produced. Commands may be freely interspersed with text, but commands and arguments must be separated from each other and from other commands and text with blanks. The input ".B.I5" is not correct--it must be written ".B .I 5" . Commas are not valid command delimiters. The case of a command or a variable name is not significant--it can be upper, lower, or mixed.

If an argument is indicated, it is usually mandatory. The only commands that permit a variable number of arguments are .TAB (or .TABS), .B, and .S . An expression can be used anywhere a numeric argument is indicated.

Some commands have an implied "break" or "return" as part of their action. These commands are indicated by flagging them with
 or <CR> in the left column in the list that follows.

For commands that set parameters or flags, the default value of that parameter or flag is indicated to the right in parentheses. In all cases, if .xxx is the command to set a flag, .NOxxx is the command to clear it.

Except for the leading period to flag commands and the leading dollar sign to flag variables, all characters that are special to RNF are under control of flags (e.g., .FLAGESC and .FLAGHYPH). In all cases, if the corresponding flag is off the character is printable.

.AGAIN

Looping command--causes RNF to reprocess the current input line or macro.

.AP (off)

Sets auto-paragraphing mode; turned off by .NOAP . With .AP on, any line with a blank in column 1 is interpreted as a .PP command to start a new paragraph. With .AP off (.NOAP), blank lines are interpreted as .B 1 commands.

.ARRAY $name size

Declares an array variable with size+1 elements, referenced as $name[i] with i in the range of 0 to size. If no

subscript appears, the reference is to the zeroeth element. An array counts as size+1 elements against the limit of 200 variables.

 .ASIS

Terminates the current line and starts "as is" mode beginning with the next input line. The mode is exited by having a line with an exclamation point (!) in column 1 (the rest of that line will be skipped). In .ASIS mode, no interpretation is done.

.ASSIGNPN variable

Assigns the "current" page number to the specified variable. When used within a figure definition, the assignment is deferred until the figure is actually printed. Also used for the same function inside a .KEEP/.ENDKEEP block. When used outside figures and .KEEP blocks, .ASSIGNPN $VAR is exactly equivalent to $VAR=$$PAGE; .

.ATITLE (off)

Turns on automatic titling, which causes chapter headings as specified in .CH commands to become the title for subsequent pages. .ATITLE is exactly equivalent to "$$ATITLE=1;" and .NOTITLE is exactly equivalent to "$$ATITLE=0;"--the variable $$ATITLE is tested as part of the default .CH macro. Default is .NOATITLE .

 .B number (1)

Writes blank lines independent of current line spacing until either number lines have been written or end-of-page is encountered. Number must be 1 or greater.

.BAR (off)

Indicates that this document will have modified sections flagged with "change bars" in the left margin. .BAR causes the document to be shifted three columns to the right to make space for the bars. The change bars themselves are turned on and off with .BB and .EB .

.BB

Turns on change bars. Must be preceded by a .BAR command. Change bars are turned off with .EB .

.BOT

Used only within the .FRCPAGE macro, .BOT puts out a bottom-of-page figure if one is pending and the footnotes scheduled

for this page, and sets the internal page size to infinite so that further output will continue on the same page.

.BR

**** This command is temporarily a DO NOTHING COMMAND **** Break -- causes the current output line (if any) to be printed without justification (but with centering, if selected).

<CR> .C

Causes the next output line to be centered between the left and right margins at the time the line is printed.

.CH text

Standard macro that does a page eject, restores the page environment, skips 12 lines, centers the word CHAPTER, leaves two blank lines, centers the text supplied in the call, and leaves three more blank lines. (See Example 7, page 70.) If .ATITLE is on, also sets the page title to the text supplied in the .CH call.

.CR

Carriage return -- causes a break followed by the number of lines selected by .SP . This command is automatically invoked after each line is justified and printed. In rare cases it can be useful to define a macro named .CR, in which case that macro will be invoked at the end of each line.

.CT

Center tab -- causes the next output word to be centered on the next tab stop.

.CTLSEQ

Starts a control sequence. As far as RNF is concerned, a "control sequence" is simply any string of characters that are not to be counted for filling and right-justification, on the assumption that they will be eaten by the output device and will not be printed. .NOCTLSEQ terminates the control sequence. Usually used for device-control such as superscripts and subscripts.

.DEF name value

Standard macro for cross-referencing. Defines a name that can be used by .REF .

<CR> .DEFFIG name

Defines a figure; terminated by .ENDFIG . Figures are called for printing by .FIG and .FIGHERE . .DEFFIG does not alter the environment and does not break the output line.

.DOT (off)

Causes all following .T, .RT, and .CT commands to fill with dots (periods) instead of spaces. A blank is always left before and after the string of dots. .NODOT cancels .DOT .

.EB

Turns off change bars (.BB turns them on).

<CR> .ELIST

Terminates the current list. Must be preceded by .LIST .

.ENDFIG

Terminates a figure definition.

.ENDKEEP

Ends a .KEEP block (see .KEEP).

.ENDNOTE

Terminates a footnote.

<CR> .ENDPAGE

Terminates the current page but does not start a new one (c.f. .PAGE, which ends the current page but also starts a new one). Used for switching header and footer formats in the middle of a document by having .ENDPAGE precede the redefinition of the .FRCPAGE macro. There is an implied .ENDPAGE at end-of-input.

.ESC

Inserts an escape character(HEX 1B) into the output file.

.F (on)

Turns on filling, which runs together input lines to make output lines that are as long as possible without exceeding the right margin.

.FIG type

Calls for a "figure" to be put out. The "type" parameter is either a name (an alphanumeric string) or a number. If it is a number, then the figure will consist of that many blank lines. If it is a name, then the figure must have been previously defined by a .DEFFIG/.ENDFIG sequence. .FIG always puts the figure at the top or bottom of a page, never in the middle-- c.f. .FIGHERE .

.FIGHERE name

Same as .FIG, except that .FIGHERE puts out the figure immediately if there is room, instead of waiting for top or bottom of page. Used when a document is nearing completion and you can tell that a figure would look better in the middle of a page instead of at the top or bottom.

.FIGLINES name $var

Following a figure definition, assigns to $var the number of lines in the figure. Used to determine line counts for positioning a figure on a page (e.g., centered).

.FLAGESC (on)

Causes the underscore character _ to mean either "literal" or "control". If .FLAGESC is on and the character after the underscore is in the range of ":" through "4", then RNF will output an ASCII control code in the range of NUL (000B) through US (037B). An ASCII ESC is thus represented as _0 . If the character following the underscore is not in that range (i.e., is a special character), then RNF simply puts out the character without attempting to interpret it. The most common use for _ is to print a character that would otherwise have some special meaning, e.g. if .FLAGSIG is on, then # must be coded in the input as _# .

.FLAGHYPH (off)

Causes the character to be interpreted as a phantom hyphen.

.FLAGOVER (on)

Causes the \ character to mean overstrike.

.FLAGSIG (on)

Causes the # character to mean significant blank.

<CR> .FLUSHFIGS

Causes all figures that have been called with .FIG and all outstanding footnotes to be printed before any more of the body of the document is output. .FLUSHFIGS is usually used at the end of a chapter, to prevent late-called figures from propagating into the beginning of the next chapter. There is an implied .FLUSHFIGS at end-of-input.

.FMT n1 n2

Formats the value of n2 according to the format code specified by n1. Both n1 and n2 may be macros or expressions. Legal values of n1 are 0 through 4, where: 0=Arabic numerals; 1=upper case alphabetic characters; 2=lower case alphabetic characters; 3=upper case Roman numerals; and 4=lower case Roman numerals.

.FOOTNOTE

Defines a footnote and queues it to appear at the bottom of the page if it will fit, or the bottom of the next if it won't go on the current page. The text of the footnote is everything between the .FOOTNOTE and the .ENDNOTE .

<CR> .HL n heading

Begins a new section with the heading specified and automatically numbers it. The n in the command must be in the range 1 to 5. Section numbers have the form i.j.k.l.m . If a .CH command preceded the .HL command, i is the chapter number; otherwise it is the number of .HL 1 commands. Within Chapter 2, the sequence .HL 1, .HL 2, .HL 3, .HL 3, .HL 2, .HL 1 produces sections numbered 2.1, 2.1.1, 2.1.1.1, 2.1.1.3, 2.1.2, and 2.2 . Three blank lines are left before each section and two blank lines follow the section header. .HL restores the page environment.

<CR> .I n

Indents the next line by n columns. The meaning of n is the same as in the .LM command.

.IBL (true)

Includes blank lines in the output file just as they are found in the input file. This command was created to take care of an ROS editor convention which requires a blank line between paragraphs. RNF considers a blank line between paragraphs as a seperate paragraph and therefore prints it as a paragraph with spacing above and below, thereby introducing 2 or more extra blank lines depending on the specified spacing. To retain the ROS spacing use .NOIBL and set

paragraph spacing to 1.

.IF expression

Executes the rest of the input line if the value of the expression is nonzero, otherwise skips the rest of the input line. Usually used within macro bodies to test conditions. If the desired conditional operations are too long to fit on one line, they can be defined as a macro and that macro called behind the .IF .

.INCLUDE filename

Allows additional text files to be included as part of the RNF input file. .INCLUDE interrupts the reading of the current file and reads the file to be included, 'filename'. After the included file is read RNF resumes reading the current file right where it left off. The .INCLUDE command can be nested to any level. The only restriction is it cannot be used recursively.

.J (on)

Sets the justification flag so that all future lines that are filled are right-justified by inserting blanks. Lines that are broken, either by .BR or by a command with an implied .BR or .CR, are never justified.

 .KEEP

Starts a block of text that is to be kept together even if it has to be printed on the next page. The block is ended with .ENDKEEP ; you do not have to count the lines. .KEEP cannot be used in the middle of a paragraph because it forces a break. .KEEP is illegal (and logically redundant) inside a figure. See also .TP .

<CR> .LE

Starts a new list element. Must be preceded by .LIST .

.LINES n (0)

Sets the physical page length to n lines. .TOP produces blank lines to fill out the page to this many lines if necessary. .LINES is not needed for line printer output. For hard-copy terminals, set n to the number of physical lines on a page (e.g., for 8 1/2 inch paper at 6 lines per inch, use n=51).

<CR> .LIST spacing [indent]

Starts a new list; a list is ended with the .ELIST command. Lists may be nested up to 5 deep. Within the list, each list element is preceded by an .LE command. The elements are numbered starting with 1 and are separated by "spacing" blank lines. Each list is indented on the left.The "ident" specification is optional. It permits the user to specify the number of spaces the list level is to be indented relative to the previous level. The default "indent" is 4 spaces.The outermost list is indented 5 + the "indent" spaces with other nested lists indented by "indent" spaces.

.LM value (1)

Resets the left margin. The value parameter can be any of the forms n, +n, or -n, or any macro or expression that evaluates to one of those forms (an expression is always unsigned). If the value is an unsigned number, then the left margin is set to that value, otherwise it is set to its current position plus the value (i.e., .LM +5 moves the margin 5 spaces right, .LM -5 moves it left).

.MACRO ...

Defines a macro. .MACRO can have any of the following forms:

```
.MACRO name = body
.MACRO name n = body
.MACRO name * = body
```

The first form defines a macro that has no parameters. The second form defines a macro that has n parameters (1<=n<=8); when the macro is invoked the next n "words" on the line are taken to be the actual parameter values. The third form defines a macro with one argument--the entire remainder of the line that calls it. The parameters are implicitly defined macros with names of name1 through name8 (or however many parameters there are). Parameter macros are initially defined as null; they are redefined when their owner macro is called. A parameter macro can be referenced at any time, even outside a call to its owner.

 .MID

Used only within the .FRCPAGE macro. .MID puts out a top-of-page figure if one is pending, then restores the output line that was saved when .FRCPAGE was invoked because of page full and resumes the body of the document.

.NMP (on)

Turns on page numbering. .NMP is exactly equivalent to "$$NMP=1;" and .NONMP is exactly equivalent to "$$NMP=0;"--the effect of turning on or off page numbering is because $$NMP is tested as part of the standard .FRCPAGE macro.

.P indent spacing test (+0 1 3)

Sets parameters for paragraphing. The "indent" parameter sets the indent for the first line of a paragraph; it can be either signed or unsigned and is interpreted the same way as the argument to .LM . The "spacing" parameter sets the number of blank lines between paragraphs (min 0). The "test" parameter sets the minimum number of lines that must remain on the page at the time the paragraph starts (i.e., an implicit .TP 2 is done at the start of each paragraph). Paragraphs themselves are delimited either by .PP commands or, if .AP is set, by lines with a blank in column 1.

<CR> .PAGE

Causes a page eject. When called outside a figure definition, .PAGE is exactly equivalent to .CR .FRCPAGE . When used inside a figure, .PAGE has no immediate effect, but causes a page eject immediately after the figure is actually printed.

.PERIOD (on)

Causes two blanks after each occurrence of the characters period (.), question mark (?), and exclamation point (!).

.PNO n

Sets the page number of the next page to n. .PNO is exactly equivalent to $$PAGE=n-1; .

<CR> .PP

Starts a new paragraph. If .AP is set, new paragraphs can also be started by a line with a blank in column 1. Note that .PP only starts a new paragraph--paragraphing parameters are set by the .P command.

.PS nlines ncolumns (57 72)

Sets page size. Sets the right margin to ncolumns (which must be an unsigned number, c.f. .RM) and causes the FRCPAGE macro to be invoked after each nlines of output is produced. Note that this "page size" does not include any footer lines generated by FRCPAGE (between .BOT and .TOP), so output pages

will contain more than nlines lines of output if a footer is present. C.f. .LINES .

.REF name placeholder

Standard macro for cross-referencing. References a name defined by .DEF . The reference can be before or after the definition. Placeholder is any string with as many characters as the corresponding value in .DEF .

.REM

Remark; causes the remainder of the input line to be ignored.

.RES

Restores the environment by popping it from the environment stack; the current environment is lost. .RES is the opposite of .SAV .

.RESENV name

Restores the environment from the specified name.

.RESPAG

Restores the "page environment", which was stored by the last .SAVPAG command or, if no .SAVPAG command has appeared, by the first page eject.

.RIGHT n (0)

Causes the entire document to be shifted right by n spaces. This command is extremely useful for producing output that is to be printed instead of examined at a terminal, because it allows you to physically shift the document without having to go through and adjust all your .LM and .RM commands. For example, if .RIGHT 10 is in effect, then .LM 1 will result in the first character being physically in column 11 of the output.

.RM value (72)

Resets the right margin. See the .LM command for syntax.

.RT

Right tab; causes the next symbol to be right-justified to the next tab stop. Used for lining up columns of numbers.

<CR> .S number

Space; similar to .B except that .S acknowledges the current spacing set by .SP .

.SAV

Saves the current environment by pushing it onto a stack (maximum of 20 environments). The opposite of .SAV is .RES .

.SAVENV name

Saves the current environment under the specified name (up to 10 different names can be used).

.SAVPAG

Stores the current environment into the "page environment", from which it can be retrieved via the .RESPAG command.

.SIG (off)

Indicates that all blanks in the input text are significant; turned off by .NOSIG . Having .SIG on is very similar to being in .ASIS mode except that commands and special characters are still interpreted. .SIG is the only way to underline in "unformatted" text, since .U will not be interpreted in .ASIS mode. When .SIG is on, command arguments must be separated from the command and from each other by exactly one blank.

.SP number (1)

Sets the interline spacing, where a value of 1 gives single spacing, 2 gives double, and so on.

.ST text

Sets the subtitle for all successive pages. .ST interfaces to the standard FRCPAGE macro; if you redefine FRCPAGE then .ST may not work.

.STANDARD

Sets up the standard enviroment for RNF operations. Standard command settings are shown below:

.AC	- off	.FLAGHYPH	- off
.AP	- off	.FLAGOVER	- on
.ATITLE	- off	.FLAGSIG	- on
.BAR	- off	.J	- on
.DOT	- off	.PEMOD	- on
.F	- on	.SIG	- off
.FLAGSC	- on	.USB	- on

.SUP

Very rarely useful; causes the current output line to be discarded (not printed).

.T

Tab--advances to the next tab stop.

.TAB c1 c2 ... c16 (no tabs)

Clears all tabs and sets new tab stops at the indicated columns.

.TABS ...

Same as .TAB

.TITLE text

Sets the title for all successive pages. .TITLE interfaces to the standard FRCPAGE macro; if you redefine FRCPAGE then .TITLE may not work.

 .TOP

Used only within the .FRCPAGE macro, .TOP finishes the previous page by writing enough lines to fill out the count specified by .LINES and starts the next line with carriage control to cause a page eject on a line printer. This first line is numbered 1, though it will usually be physically line 4 on the page.

.TP n

Test page--if fewer than n lines remain on the page (allowing for spacing set by .SP), then do a page eject, otherwise do nothing. .TP does not break the current line--the page eject is deferred until the current line is finished.

.U

Starts underlining, which is turned off by .NOU . All characters output between the .U and .NOU commands will be underlined except for blanks. Significant blanks will be underlined if and only if .USB is on.

.USB (on)

Causes significant blanks to be underlined when .U is on. This is the default; .NOUSB turns it off.

.VAR $name or .VAR $name = value

Declares a single-valued variable which can be used in subsequent expressions. The value is initialized to zero in the first form. In the second form, value can be any expression or macro that evaluates to an expression.

<CR> .VT n

Vertical tab--issues enough blank lines to make the next output be on line n.

.X

Extend--causes the first character output after the .X to immediately follow the last character output before the .X . That is, it concatenates the word preceeding the .X with the word following the .X . An outstanding .X can be canceled by .NOX (usually after a macro sets .X when you don't want it).

(This page is intentionally blank.)

Examples

The following examples illustrate most of RNF's commands and all of its major features. They are arranged roughly in order of difficulty--Examples 1 through 3 will show you enough to use RNF for simple applications.

In most of these examples, an RNF input file is presented followed by the corresponding RNF output. The format of the examples may be somewhat distracting. Please note carefully that the text of each example is, in fact, an explanation of what the example is illustrating. This means that you are reading what is being operated on by RNF. (Actually, this entire report is written in RNF.) You should, therefore, read the "RNF OUTPUT" sections of the examples and refer only to the preceeding input to see how the RNF commands are used. The RNF output(which you are reading) may then be referred to in order to see the effect of the RNF command. The examples are intended as comprehensive, detailed reference manual for someone who is actually writing RNF code and will probably represent overkill to the person who simply wants to determine the features and capabilities of RNF. All examples were prepared on ASCII terminals at an M68000 machine with a ROS operating system.

List of Examples

Example 1: Common Commands

```
.LM 1 .RM 65 .SP 1 .P +3 1 3
.C RNF .C Input file
.PP A RNF input file consists of a sequence of words
and commands separated by blanks.  A "word" is any sequence
characters that does not contain a blank.  Any word that
starts with a period and is not a number is a command.
For example, _.I is a command but .234 is not.  (A word that starts
with underscore/period is not a command.)
.NOJ .PP
RNF gathers words together into
lines, which start at the left margin unless they start a paragraph
or are indented with a _.I command.  Normally, RNF justifies lines by
inserting blanks, but this can be turned off with a _.NOJ command.
To insert tables or examples, the simplest way is to use "asis" text:
.ASIS

     1. Precede the table with an .ASIS command.
     2. Type the table exactly as you want it to appear in
          the output.  RNF does not reformat it at all.
     3. Follow the table with a line containing just an
          exclamation point in column 1.


********** RNF OUTPUT STARTS ON NEXT PAGE
```

Example 1 (continued)

RNF
Input file

A RNF input file consists of a sequence of words and commands separated by blanks. A "word" is any sequence characters that does not contain a blank. Any word that starts with a period and is not a number is a command. For example, .I is a command but .234 is not. (A word that starts with underscore/period is not a command.)

RNF gathers words together into lines, which start at the left margin unless they start a paragraph or are indented with a .I command. Normally, RNF justifies lines by inserting blanks, but this can be turned off with a .NOJ command. To insert tables or examples, the simplest way is to use "asis" text:

1. Precede the table with an .ASIS command.
2. Type the table exactly as you want it to appear in the output. RNF does not reformat it at all.
3. Follow the table with a line containing just an exclamation point in column 1.

Example 2: Indention and Significant Blanks

```
.P +0 1 3
.PP  To indent a single line, use the _.I command:

.I +5    This sentence is indented 5 spaces.
.PP When using _.I, don't forget the plus sign (+) in front of the
number--if the plus sign isn't there the number will be treated
as a column number instead of an indent.
.PP  There are two easy ways to get blank lines in the output:

.I +5   1.#Use the _.B or _.S commands.
.I +5   2.#Put a blank line in the input (assuming that you are
.I +5   #####not auto-paragraphing).
.B 5
One easy way to force blank characters in the output is by
using the "significant blank" character, the pound sign (_#).
The pound character acts like any other character during filling
and justification, but is replaced by a blank just before the
line is printed.
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 2 (continued)

To indent a single line, use the .I command:

.I +5
This sentence is indented 5 spaces.

When using .I, don't forget the plus sign (+) in front of the number--if the plus sign isn't there the number will be treated as a column number instead of an indent.

There are two easy ways to get blank lines in the output:

1. Use the .B or .S commands.
2. Put a blank line in the input (assuming that you are not auto-paragraphing).

One easy way to force blank characters in the output is by using the "significant blank" character, the pound sign (#). The pound character acts like any other character during filling and justification, but is replaced by a blank just before the line is printed.

Example 3: Paragraphing

```
.P +5 1 2
.PP The paragraphing facility in RNF enables fast, convenient
text entry in a natural format.
.PP Paragraphing is set up by the _.P command, which has the
format:
.B .I 9 _.P indent spacing test
.PP Indent specifies how the first line of each paragraph is to
be indented.  Usually indent will be signed (+5), in which case
it is relative to the left margin at the time the paragraph
starts.  If indent is unsigned (5), it is the column number in
which the first line should start.  Spacing specifies the number
of spaces between paragraphs and is dependent on the overall
spacing set by _.SP#.  Test specifies the minimum number of
lines that must be left on a page to start a new paragraph.
.SAV
.LM +10 .RM -5 .SP 2
.PP Each paragraph is started with a _.PP command.
Notice how this indented paragraph still has a 5-space indent
as a result of the _.P command.  Also, notice that it is
double-spaced.
You can change margins and spacing
as often as you like with the _.P and _.SP commands, such as to
insert a single-spaced indented quote into a double-spaced thesis.
.P -8 1 2 .SP 1
.PP This paragraph is done with a "hanging indent", in which the
first line of the paragraph sticks out to the left of the body
of the paragraph.
.RES .AP
 As an alternative to starting each paragraph with an explicit _.PP
command, you can also set "autoparagraphing" mode with the _.AP
command.  In autoparagraphing mode, any blank line or line with
a blank in its first column starts a new paragraph, just as if it
were a _.PP command.
 Be careful that you do not confuse the _.PP and _.P commands.
_.PP is used to start each new paragraph (assuming that you
are not autoparagraphing) and _.P is used to establish
the format of all successive paragraphs until another
_.P command is encountered.
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 3 (continued)

The paragraphing facility in RNF enables fast, convenient text entry in a natural format.

Paragraphing is set up by the .P command, which has the format:

.P indent spacing test

Indent specifies how the first line of each paragraph is to be indented. Usually indent will be signed (+5), in which case it is relative to the left margin at the time the paragraph starts. If indent is unsigned (5), it is the column number in which the first line should start. Spacing specifies the number of spaces between paragraphs and is dependent on the overall spacing set by .SP . Test specifies the minimum number of lines that must be left on a page to start a new paragraph.

Each paragraph is started with a .PP command. Notice how this indented paragraph still has a 5-space indent as a result of the .P command. Also, notice that it is double-spaced. You can change margins and spacing as often as you like with the .P and .SP commands, such as to insert a single-spaced indented quote into a double-spaced thesis.

This paragraph is done with a "hanging indent", in which the first line of the paragraph sticks out to the left of the body of the paragraph.

As an alternative to starting each paragraph with an explicit .PP command, you can also set "autoparagraphing" mode with the .AP command. In autoparagraphing mode, any blank line or line with a blank in its first column starts a new paragraph, just as if it were a .PP command.

Be careful that you do not confuse the .PP and .P commands. .PP is used to start each new paragraph (assuming that you are not autoparagraphing) and .P is used to establish the format of all successive paragraphs until another .P command is encountered.

Example 4: Tab Stops

```
.DEF RTABS $$PAGE  .rem  cg- used in following example.PP
RNF tab stops are set with the _.TABS command, which can also be
written as _.TAB#.  The format is:

.I +5 _.TABS c1 c2 _._._. c15

.TP 2
where c1 through c15 are character positions, in ascending order,
where the tab stops are to be set.  All previous tab stops are
cleared by _.TABS, so you must set all stops simultaneously.
There are three commands for using tabs: _.T, _.RT, and _.CT#.
_.T is similar to the tab key on a typewriter--it causes the
next word to begin at the next tab stop.  _.RT (right tab)
causes the next word to end at the next stop, and _.CT (center tab)
causes it to be centered on the next tab stop.  To illustrate:

.TABS 15 30 45  .T * .T * .T * .BR
These are .T normal .CT centered .RT and#right tabs.
.PP
The following feature should be noted:
.SAV .LM 30 .TAB 30 .P -25 1 5 .DOT
.PP Justification .T Having tabs in a line does not suppress
justification.  Only the text after the last tab in a line is blank
padded, which is very convenient for hanging indention.  Notice
also the use of _.DOT to fill the tabbed-over space with periods.
.RES
.PP .KEEP
For tabular data, it is usually most convenient to turn off
filling with a _.NOF command.  For example:

.SAV .NOF .LM 10 .TABS 25 38 53
Item .CT Price# .CT Quan.Sold .CT Total#Price####
---- .CT -----# .CT --------- .CT -----------####
Worms .RT .79 .RT 6 .RT 4.74
Eggs .RT 1.39 .RT 10 .RT 13.90
Fish .RT 3.19 .RT 76 .RT 242.44
.RES .ENDKEEP
.PP Notice also the use of _.KEEP_/.ENDKEEP to surround the table.
This guarantees that the table will not be broken across two pages.
.PP
To right-tab or center-tab
a word constructed of several input words "stuck
together" with the _.X command, the _.RT or _.CT command
must come just before the last input word.  For example:

.TAB 60 .T * .BR This .X is .X stuck .X .RT together.
.PP
If you use _.DOT a lot, you may need to know one subtlety of
tab stops: a tab movement is not actually done until the next
word is added to the output line.  This means that the
```

Example 4 (continued)

"tabbed-over" field will not be filled with periods unless there is actually another word after it. This word may, however, be a significant blank (_#), in which case the output will appear as if there is no word following the dots. For example, with tab stops set at columns 10 and 50, the following RNF input

```
.TAB 10 50 .NOF
.I +5 _.T  before  _.DOT  _.T  after  _.NODOT
.I +5 _.T  before  _.DOT  _.T         _.NODOT
.I +5 _.T  before  _.DOT  _.T _#      _.NODOT
.I +5 _.T  tabs    _.DOT  _.T _# _.REF RTABS ?? _.NODOT
```

will produce

```
.T  before  .DOT  .T  after  .NODOT
.T  before  .DOT  .T         .NODOT
.T  before  .DOT  .T  #      .NODOT
.T  tabs    .DOT  .T  # .REF RTABS ?? .NODOT
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 4 (continued)

RNF tab stops are set with the .TABS command, which can also be written as .TAB . The format is:

```
.TABS c1 c2 ... c15
```

where c1 through c15 are character positions, in ascending order, where the tab stops are to be set. All previous tab stops are cleared by .TABS, so you must set all stops simultaneously. There are three commands for using tabs: .T, .RT, and .CT . .T is similar to the tab key on a typewriter--it causes the next word to begin at the next tab stop. .RT (right tab) causes the next word to end at the next stop, and .CT (center tab) causes it to be centered on the next tab stop. To illustrate:

```
            *             *             *
These are   normal     centered   and right tabs.
```

The following feature should be noted:

Justification ..... Having tabs in a line does not suppress justification. Only the text after the last tab in a line is blank padded, which is very convenient for hanging indention. Notice also the use of .DOT to fill the tabbed-over space with periods.

For tabular data, it is usually most convenient to turn off filling with a .NOF command. For example:

Item	Price	Quan.Sold	Total Price
Worms	.79	6	4.74
Eggs	1.39	10	13.90
Fish	3.19	76	242.44

Notice also the use of .KEEP/.ENDKEEP to surround the table. This guarantees that the table will not be broken across two pages.

To right-tab or center-tab a word constructed of several input words "stuck together" with the .X command, the .RT or .CT command must come just before the last input word. For example:

```
                                                *
                                        Thisisstucktogether.
```

If you use .DOT a lot, you may need to know one subtlety of tab stops: a tab movement is not actually done until the next word is added to the output line. This means that the "tabbed-over" field will not be filled with periods unless there is actually another

Example 4 (continued)

word after it. This word may, however, be a significant blank (#), in which case the output will appear as if there is no word following the dots. For example, with tab stops set at columns 10 and 50, the following RNF input

```
     .T before .DOT .T after .NODOT
     .T before .DOT .T .NODOT
     .T before .DOT .T # .NODOT
     .T tabs   .DOT .T # .REF RTABS ??  .NODOT
```

will produce

```
        before ..................................... after
        before
        before .....................................
        tabs   .....................................   63
```

Example 5: Paging Controls

```
.NMP
.TITLE RNF Paging Controls .TAB 34 .T .U Example .NOU
.ST .B 1 (An illustration for the manual)
.PAGE
.C Paging Controls
.C ---------------
.PP The standard page format consists of a 1-inch top margin, a title
line with a page number, a subtitle line, two blank lines, and then
your text.  By default the title and subtitle are blank.  You can
fill in the title and subtitle lines with the following commands:

.I +5    _.TITLE  anything
.I +5    _.ST anything

where "anything" means any combination of text and RNF commands.
Notice how the title defined for this example contains tab commands,
while the subtitle even generates an extra blank line.
.PP
The page number is normally present but can be turned off with
the _.NONMP command.  If you want the page number in a different
position or, for example, in Roman numerals, then you have to
write a FRCPAGE macro--see the Table of Contents to find the
appropriate section.
.PP
Once you have defined a title and subtitle, RNF will automatically
put them on each page.  RNF advances to a new page when the page
is full or when you tell it to with a _.PAGE command.  The
length of a page is set with the _.PS command.
.PP
Frequently you will
want to guarantee that blocks of text, such as a table or an
example with its introduction, do not split across page boundaries.
The easiest way to do this is with _.KEEP and _.ENDKEEP#.  Simply
put a _.KEEP command before the text you want kept together and
an _.ENDKEEP after it--no counting lines!  The _.TP command
can also be used if you don't mind counting lines.  In the
middle of a paragraph, _.TP is your only option because _.KEEP
causes a "break".  See Example#4 on Tabs for an example of _.KEEP#.
.PP
Other paging controls, much more exotic, are provided by the
figure and footnote facilities.  See the Table of Contents.


********** RNF OUTPUT STARTS ON NEXT PAGE
```

(An illustration for the Manual)

Paging Controls

The standard page format consists of a 1-inch top margin, a title line with a page number, a subtitle line, two blank lines, and then your text. By default the title and subtitle are blank. You can fill in the title and subtitle lines with the following commands:

```
.TITLE anything
.ST anything
```

where "anything" means any combination of text and RNF commands. Notice how the title defined for this example contains tab commands, while the subtitle even generates an extra blank line.

The page number is normally present but can be turned off with the .NONMP command. If you want the page number in a different position or, for example, in Roman numerals, then you have to write a FRCPAGE macro--see the Table of Contents to find the appropriate section.

Once you have defined a title and subtitle, RNF will automatically put them on each page. RNF advances to a new page when the page is full or when you tell it to with a .PAGE command. The length of a page is set with the .PS command.

Frequently you will want to guarantee that blocks of text, such as a table or an example with its introduction, do not split across page boundaries. The easiest way to do this is with .KEEP and .ENDKEEP . Simply put a .KEEP command before the text you want kept together and an .ENDKEEP after it--no counting lines! The .TP command can also be used if you don't mind counting lines. In the middle of a paragraph, .TP is your only option because .KEEP causes a "break". See Example 4 on Tabs for an example of .KEEP .

Other paging controls, much more exotic, are provided by the figure and footnote facilities. See the Table of Contents.

Example 6: Underlining, Overstriking, and Hyphenation

```
.FLAGHYPH
.P +5 0 3 .SP 2
.PP Underlining is turned on with the _.U command and turned off
with _.NOU#.  For example, .U this stuff is underlined .NOU and
this isn't.  Punctuation .U will be underlined also, .NOU (note
the preceding comma) so you may want to use _.X to concatenate
the punctuation after turning underlining .U off .NOU .X .  (Note
the period.)  Significant .U bl##anks .NOU are ordi/narily
under/lined, but you can turn this off .NOUSB with the _.NOUSB
.U com###mand .NOU .X .  Note the use of phantom hyphens to
syl/lab/icate un/com/fort/ably long words, after turning on
hyphenation with the _.FLAGHYPH command.
.PP .SP 1
Arbitrary characters can be "overstruck", or printed at the same
position, by using the overstrike character backslash (_\).
For example, a tolerable O\/ can be produced by overstriking an "O"
with a "_/".  Overstriking is limited to four characters at a single
posi/tion (or three characters and an underscore).
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 6 (continued)

Underlining is turned on with the .U command and turned off with .NOU . For example, this stuff is underlined and this isn't. punctuation will be underlined also, (note the preceding comma) so you may want to use .X to concatenate the punctuation after turning underlining off. (Note the period.) Significant bl anks are ordinarily underlined, but you can turn this off with the .NOUSB command. Note the use of phantom hyphens to syllabicate uncomfortably long words, after turning on hyphenation with the .FLAGHYPH command.

Arbitrary characters can be "overstruck", or printed at the same position, by using the overstrike character backslash (\). For example, a tolerable Ø can be produced by overstriking an "O" with a "/". Overstriking is limited to four characters at a single position (or three characters and an underscore).

Example 7: Header Levels and Chapters (.HL and .CH)

```
.AP
.CH Standard Chapter and Section Headers
.HL 1 PURPOSE
 The purpose of this section is to demonstrate the effect of
the standard _.CH and _.HL commands.
.HL 1 HOW TO USE H\_EADER L\_EVELS (_.HL)
 The _.HL command enables the typist to organize text into
sections, and sections within sections, with sequential numbers
which are changed by the computer if sections are added or
deleted.
.HL 2 Format of the Command
 The format of the _.HL command is:

.I +5 _.HL number text
 The number is the level number, where 1 is the outermost
level, 2 is the next level, and so on.  The text is the title
of the section.
.HL 3 Effect of the Command
 The _.HL command starts a section at the level specified and
uses text as the header.  The number can be in the range of
1 to 5.  The section number output is in the form i.j.k.l.m#.
If a _.CH command has been used, i is the chapter number;
otherwise it is the number of _.HL#1 commands so far.
.HL 3 Format of the Output
 The command does a _.CR, a _.TP 9, a _.RESPAG, and a _.B#3#.
Then it prints the section number and the section title (the
text).  Two blank lines follow the header line.
.HL 2 Numbering Convention
 Each time levels are nested (for example, _.HL 1 followed by
_.HL#2), the numbering is restarted at 1 for the higher numbered
level.  Successive _.HL commands at the same level increment
the section by 1 in the last position.
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 7 (continued)

CHAPTER 1

Standard Chapter and Section Headers

1.1 PURPOSE

The purpose of this section is to demonstrate the effect of the standard .CH and .HL commands.

1.2 HOW TO USE HEADER LEVELS (.HL)

The .HL command enables the typist to organize text into sections, and sections within sections, with sequential numbers which are changed by the computer if sections are added or deleted.

1.2.1 Format of the Command

The format of the .HL command is:

```
.HL number text
```

The number is the level number, where 1 is the outermost level, 2 is the next level, and so on. The text is the title of the section.

Example 7 (continued)

1.2.1.1 Effect of the Command

The .HL command starts a section at the level specified and uses text as the header. The number can be in the range of 1 to 5. The section number output is in the form i.j.k.l.m . If a .CH command has been used, i is the chapter number; otherwise it is the number of .HL 1 commands so far.

1.2.1.2 Format of the Output

The command does a .CR, a .TP 9, a .RESPAG, and a .B 3 . Then it prints the section number and the section title (the text). Two blank lines follow the header line.

1.2.2 Numbering Convention

Each time levels are nested (for example, .HL 1 followed by .HL 2), the numbering is restarted at 1 for the higher numbered level. Successive .HL commands at the same level increment the section by 1 in the last position.

(This page is intentionally blank.)

Example 8: Lists (.LIST, .LE, and .ELIST)

```
.C Using Lists
.P +0 1 3
.AP
 Automatically numbered lists can be generated by the _.LIST,
_.LE, and _.ELIST commands.  The _.LIST command has the format:
 .I 9 _.LIST spacing
 where spacing is a number in the range 1 through 5 which
specifies the spacing to be used within the list.
 The list is generated as follows:
.LIST 1
.LE Each list element is preceded by the _.LE command.
.LE List elements are separated by "spacing" blank lines.
.LE Each element is automatically numbered beginning at
1 and incremented by 1.
.LE The list is terminated by the _.ELIST command (which
also resets the margins and spacing).
.LE Lists can be nested within lists.  This is handled
as follows:
.LIST 1
.LE The left margin is set 9 columns to the right for
the outermost list, and 4 columns to the right for each
inner list.
.LE Each _.LIST must be terminated by a matching _.ELIST.
The _.ELIST resets the margins and spacing as they were
before the _.LIST command.
.LE For users of autoparagraphing, the paragraph margin
is set to coincide with the left margin, and paragraph
spacing is set to the list spacing.
 This means that autoparagraphing can be conveniently
used at any list level.
 The paragraph values are reset by the _.ELIST command.
.ELIST
.LE Lists can be nested up to 5 levels.
.ELIST
 _.LIST cannot produce lists numbered in Roman numerals or
alphabetically, but macros that can are not too difficult
to write--see Example 10.
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 8 (continued)

Using Lists

Automatically numbered lists can be generated by the .LIST, .LE, and .ELIST commands. The .LIST command has the format:

```
.LIST spacing
```

where spacing is a number in the range 1 through 5 which specifies the spacing to be used within the list.

The list is generated as follows:

1. Each list element is preceded by the .LE command.

2. List elements are separated by "spacing" blank lines.

3. Each element is automatically numbered beginning at 1 and incremented by 1.

4. The list is terminated by the .ELIST command (which also resets the margins and spacing).

5. Lists can be nested within lists. This is handled as follows:

    1. The left margin is set 9 columns to the right for the outermost list, and 4 columns to the right for each inner list.

    2. Each .LIST must be terminated by a matching .ELIST. The .ELIST resets the margins and spacing as they were before the .LIST command.

    3. For users of autoparagraphing, the paragraph margin is set to coincide with the left margin, and paragraph spacing is set to the list spacing.

       This means that autoparagraphing can be conveniently used at any list level.

       The paragraph values are reset by the .ELIST command.

6. Lists can be nested up to 5 levels.

.LIST cannot produce lists numbered in Roman numerals or alphabetically, but macros that can are not too difficult to write--see Example 10.

Example 9: Some Macros

This example illustrates a couple of simple and useful macros with parameters. Macro D is typical of those used for section headers,document lists, command summaries, etc. Macro HI is used throughout this manual-- the only reason it is not in the standard set is that its exact form depends on the type of paragraphs you like to use.

```
.MACRO D 2 = .RESENV DENV .B .TP 4 .D1 .RT .D2 .B .LM +4 .RM -4
.MACRO HI 2 = .SAV .LM .HI1 .P $$LM 1 2 .LM .HI2 .TAB $$LM
.PP A little explanation of these macros may be needed. Macro HI
is used for establishing hanging indents. It is called as follows:

.I +4 _.HI indent1 indent2

where:
.HI +5 +10
.PP indent1 .T indicates where the first line of each paragraph
of the hanging indent should start. It follows the conventions
for _.LM, in that it can be either signed or unsigned. If indent1
is unsigned, it is an absolute column number; if it is signed,
then the position is relative to the left margin at the time
HI is invoked.
.PP indent2 .T indicates where succeeding lines of the hanging
indent should start. Like indent1, indent2 can be either signed
or unsigned. If indent2 is signed, then the indicated position
is relative to the position set by indent1.
.RES
.PP The most subtle aspect of HI is probably the use of _$$LM as
an argument to _.P and _.TAB#. Remember that _$$LM is a predefined
variable that is the current left margin and is set by the _.LM
command.
.PP To use macro D, it is first necessary to set up the proper
envi/ron/ment, named DENV. The rest of its use is fairly straight/
forward. For example, consider the following:

.SAV .LM 4 .RM 66 .TAB $$RM .DOT .SAVENV DENV .NODOT
     #              .RT   Doc.#Number
Document#Name       .RT    (if#any)#
-------------       .RT   ----------

.D RNF#Users#Manual TM#86327
   M68000 users manual for RNF, the text formatter for the M68000.


.RES
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 9 (continued)

A little explanation of these macros may be needed. Macro HI is used for establishing hanging indents. It is called as follows:

```
.HI indent1 indent2
```

where:

indent1 indicates where the first line of each paragraph of the hanging indent should start. It follows the conventions for .LM, in that it can be either signed or unsigned. If indent1 is unsigned, it is an absolute column number; if it is signed, then the position is relative to the left margin at the time HI is invoked.

indent2 indicates where succeeding lines of the hanging indent should start. Like indent1, indent2 can be either signed or unsigned. If indent2 is signed, then the indicated position is relative to the position set by indent1.

The most subtle aspect of HI is probably the use of $$LM as an argument to .P and .TAB . Remember that $$LM is a predefined variable that is the current left margin and is set by the .LM command.

To use macro D, it is first necessary to set up the proper environment, named DENV. The rest of its use is fairly straightforward. For example, consider the following:

Document Name	RNF Number (if any)
RNF Users Manual	TM 86237

M68000 users manual for RNF, the text formatter for the M68000.

Example 10: More Macros

```
.ARRAY $AL 5
.ARRAY $ALF 5
.MACRO FLIST 1 = .FLISTA .FLISTB
.MACRO FLISTA = .SAV .LM +6 .VAR $ALTEMP = .FLIST1
.MACRO FLISTB = $AL=$AL+1; $AL[$AL]=0; $ALF[$AL]=$ALTEMP; .TAB $$LM-3
.MACRO FLE = .B .I 1 .RT .FMT $ALF[$AL] $AL[$AL]=$AL[$AL]+1 .X _.# .X
.MACRO FELIST = .RES $AL=$AL-1;
.P +0 1 3 .TP 20
.PP This example shows some fairly complicated macros that
are very easy to use.  They implement some new list commands that
are similar to the standard _.LIST, _.LE, and _.ELIST commands (see
Example 8), but that can number lists in Arabic numerals,
alphabetically, or in Roman numerals.  They work as follows:
.FLIST 1 .REM (set for upper-case letters)
.FLE The _.FLIST macro does the following:
.FLIST 4 .REM (set for lower-case Roman numerals)
.FLE The environment is saved with _.SAV#.
.FLE A _.LM command is used to indent the left margin.
.FLE The value of the _.FLIST argument is transferred to a
temporary numeric variable, _$ALTEMP#.
.FLE The list level is incremented, the count for this level is
set to zero, and the format of the indexing is saved.
.FELIST
.FLE Then, each invocation of _.FLE does the following:
.FLIST 2 .REM (set for lower-case letters)
.FLE A blank line is issued and _.I is used to start the output
line at column 1 so that the list number can be right-tabbed
to line up with the left margin.
.FLE _.FMT is called to output the current list number in the
selected format.
.PP The number is incremented in the process as a result of
evaluating the expression _$AL[$AL]=$AL[$AL]+1;#.
.FLE A period and one significant blank are added to the
"number".
.FELIST
.FLE The _.FELIST macro has it easy--all it has to do is _.REStore
the previous environment and decrement the list level.
.FELIST
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 10 (continued)

This example shows some fairly complicated macros that are very easy to use. They implement some new list commands that are similar to the standard .LIST, .LE, and .ELIST commands (see Example 8), but that can number lists in Arabic numerals, alphabetically, or in Roman numerals. They work as follows:

A. The .FLIST macro does the following:

   i. The environment is saved with .SAV .

   ii. A .LM command is used to indent the left margin.

   iii. The value of the .FLIST argument is transferred to a temporary numeric variable, $ALTEMP .

   iv. The list level is incremented, the count for this level is set to zero, and the format of the indexing is saved.

B. Then, each invocation of .FLE does the following:

   a. A blank line is issued and .I is used to start the output line at column 1 so that the list number can be right-tabbed to line up with the left margin.

   b. .FMT is called to output the current list number in the selected format.

      The number is incremented in the process as a result of evaluating the expression $AL[$AL]=$AL[$AL]+1; .

   c. A period and one significant blank are added to the "number".

C. The .FELIST macro has it easy--all it has to do is .REStore the previous environment and decrement the list level.

Example 11: Macros For This Manual

In this example are listed all the major macros that were used to format this manual. These macros are intended to give experienced RNF users some ideas about what is possible. Do not bother to read this example unless you are experienced with RNF, and even then do not expect to understand it on the first try--these macros are even more complicated and subtle than the ones that implement the standard page format. The results of these macros are not shown as part of this example because they are abundantly illustrated in the remainder of the manual.

The following macros handle the page formatting. They are very similar to the versions described in the narrative on FRCPAGE (page 21). Notice how macros CHECKODD and CHECKEVEN are used to determine whether the page number is odd or even, and how .RIGHT is used by macro XTOP (via XTOPA and XTOPB) to set different margins for left and right pages, to make printing and binding easier.

```
.rem latest revision date
.MACRO REVISED = September#1984
.rem margin shifts for even and odd pages
.VAR $EVENRIGHT = 6 .VAR $ODDRIGHT = 10
.rem page size in lines and right margin
.VAR $PS = 57 .VAR $RM = 68
.rem even/odd page evaluating functions
.VAR $TRUE
.MACRO CHECKODD = $TRUE=($$PAGE-($$PAGE/2)*2.NE.0);
.MACRO CHECKEVEN = $TRUE=($$PAGE-($$PAGE/2)*2.EQ.0);
.rem page formatting macros
.MACRO FRCPAGE = .BOT .FOOT .XTOP .HEAD .MID
.MACRO XTOP = .BR .TOP $$PAGE=$$PAGE+1; .XTOPA .XTOPB
.MACRO XTOPA = .CHECKEVEN .IF $TRUE .RIGHT $EVENRIGHT
.MACRO XTOPB = .CHECKODD  .IF $TRUE .RIGHT $ODDRIGHT
.MACRO FOOT = .SAV .RESPAG .FOOTA .RES
.MACRO FOOTA = .IF $$PAGE.NE.0 .B 2 .LEFTPN .RIGHTPN
.MACRO LEFTPN = .CHECKEVEN .IF $TRUE $$PAGE
.MACRO RIGHTPN = .CHECKODD .IF $TRUE .RT $$PAGE
.MACRO HEAD = .SAV .RESPAG .HEADA .HEADB .RES
.MACRO HEADA = NASA Langley Research Center .RT RNF .BR
.MACRO HEADB = Software Development Lab .RT .REVISED .B .SST1 .B
.MACRO SST * =
```

Macros SEC and MAIN handle the headers for each main section and subsection.

```
.MACRO SEC * = .RESPAG .B 4 .TP 10 .SEC1 .BR
.MACRO MAIN = .STARTONODD .RESPAG # .B 2
.MACRO STARTONODD = .ENDPAGE .CHECKODD .IF $TRUE .BLANKPG
.MACRO BLANKPG = .VT 25 .C (This page is intentionally blank.) .PAGE
```

Example 11 (continued)

Macro EXAMPLE is used to head all the examples. Notice how it uses SST to interface with FRCPAGE to produce the "Example n (continued)" header lines on continuation pages. Also notice how each call to EXAMPLE causes a name EXAMPLEn to be .DEFined for cross-referencing.

```
.MACRO EXAMPLE 2 = .EX1 .C Example .EXAMPLE1 .X : .EXAMPLE2 .EX4
.MACRO EX1 = .BR .CLRSST .PAGE .EX2 .RESPAG .NOAP
.MACRO EX2 = .SST .C Example .EXAMPLE1 (continued) .B
.MACRO EX3 = EXAMPLE .X .EXAMPLE1
.MACRO EX4 = .DEF .EX3 $$PAGE .B 2 .ASIS
.MACRO CLRSST = .SST
```

Sample call: .EXAMPLE 1 Common#Commands

SECDESC is used in the Introduction to format the listing of the 10 sections of the manual. SPC and XREF are used in Special Characters and Flags to format the table of special characters, commands, and column numbers

```
.MACRO SECDESC * = .B 1 .TP 4 .I 2 .SECDESC1 .P +0 0 2 .LM 7 .I -2
.MACRO SPC 4 = .BR .CT .SPC1 .CT .SPC2 .CT .SPC3 .CT .SPC4 .T
.MACRO XREF 3 = .X .XREF1 .X .REF .XREF2 .XREF3
```

Sample call:

```
.SPC _\ backslash overstrike _.FLAGOVER .XREF # EXAMPLE7 ??
```

Macro CMD is used to format the List of Commands.

```
.MACRO CMD 3 = .CMDA .CMDB
.MACRO CMDA  = .BR .LM 1 .B 1 .TP 4 .TAB 6 $$RM .CMD2
.MACRO CMDB  = .T .CMD1 .RT .CMD3 .LM 8 .B 1
```

Sample calls:

```
      .CMD _.KEEP <BR>
  and .CMD _.P#indent#spacing#test # (+0#1#3)
```

Example 12: Figures

```
.PAGE
.DEFFIG FIGFEAT
.SAV .LM 8 .RM 64 .P -3 1 1 .SAVENV FIGENV .B 2
.PP  1.#The whole text of a figure is guaranteed to appear on one
       page.  This restricts the maximum size of a figure to
       somewhat less than the page size.
.PP  2.#The surrounding document text "flows" around the figure,
       eliminating excessive white space that would result from
       just doing a page eject before the figure.
.B 1 .C .U Figure 1_. .NOU Features of Figures .B 2 .RES
.ENDFIG
.DEFFIG FIGUREDC
.SAV .RESENV FIGENV .B 2
.PP  1.#Any lasting environment changes made during the definition
       will affect text after the definition, not after the call.
.PP  2.#References to the page number of a figure must be made
       with the _.ASSIGNPN command.
.PP  3.#When the figure definition starts, the output line number
       (variable _$$OLNO) is set to 1, so that any line number
       reference will be relative to the start of the figure, not
       the start of the page on which it is printed.
.B 1 .C .U Figure 2_. .NOU Aspects of Figure Definition_/Call
.B 2 .RES
.ENDFIG
.DEFFIG FIGRSTRCT
.SAV .RESENV FIGENV .B 2
.PP  1.#You can have a maximum of 200 figures and footnotes
       defined at any one time.  It is common to define all
       figures at the start of the document and then just call
       them where they are needed.  A figure becomes undefined
       when it is printed, so any number of figures can be
       handled by judicious interspersing of definitions and
       calls.
.PP  2.#The maximum number of "outstanding" figures is 20.
       An outstanding figure is one that has been called but
       has not yet been printed.
.B 1 .C .U Figure 3_. .NOU Restrictions on Figures .B 2 .RES
.ENDFIG
.PP
In this example of figures, note how the figures are defined
and called in two separate steps.  This makes it very easy to
move the figure around in the final document, because the command
that calls the figure is only one line.  Notice how the text
outside of the figures "flows" around them to close
up all the blank space.  The small amount of blank space before and
after each figure is provided with _.B commands within the figure
definition.
.REM Now seems like a good time to call the figures....
.FIG FIGFEAT .FIG FIGUREDC .FIG FIGRSTRCT
.PP
```

Example 12 (continued)

The _.DEFFIG and _.ENDFIG commands have been previously described, as has the form of _.FIG used in this example. There is an alternate form of _.FIG that is useful if you merely want to reserve space for subsequent cut-and-paste. That form is:

```
.I +5 _.FIG n
```

where n is the number of lines of blank space desired (independent of spacing). In this case no explicit figure definition is needed.

```
.PP
```

If you do not like RNF's placement of a figure, you can override it via the _.FIGHERE command. It has the same syntax as _.FIG; the difference is that _.FIGHERE causes the figure to be printed immediately if there is space on the page and no other figures are queued. If other figures are queued or you are too near end-of-page, _.FIGHERE converts to _.FIG#.

```
.PP
```

Outside figures, you can access the page number by referring to the predefined variable _$$PAGE.
This trick will not work for figures, however, because it will get the page number current at the time the figure is defined, not the page number on which the figure is eventually printed. To circumvent this problem, there is a special _.ASSIGNPN command, which has the syntax:

```
.I +5 _.ASSIGNPN _$var
```

The variable _$var must have been previously declared. When _.ASSIGNPN is used outside a figure definition, it is equivalent to "$var=$$PAGE;", but when used inside a figure definition, the assignment is deferred until the figure is actually printed.

```
.PP
```

There are two other special commands for use with figures:

```
.I +5 _.FIGLINES figurename _$var
.I +1 and _.FLUSHFIGS
```

_.FIGLINES will assign to _$var the number of lines in the indicated figure. It can be used to determine how many blank lines are needed to center a figure on a page, for example. _.FLUSHFIGS is used at the end of a section to guarantee that any figures called within that section have been printed. (Otherwise, figures from chapter 3 can migrate into chapter 4, which looks terrible.) There is an implied _.FLUSHFIGS command at the end of the document.

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 12 (continued)

In this example of figures, note how the figures are defined and called in two separate steps. This makes it very easy to move the figure around in the final document, because the command that calls the figure is only one line. Notice how the text outside of the figures "flows" around them to close up all the blank space. The small amount of blank space before and after each figure is provided with .B commands within the figure definition.

The .DEFFIG and .ENDFIG commands have been previously described, as has the form of .FIG used in this example. There is an alternate form of .FIG that is useful if you merely want to reserve space for subsequent cut-and-paste. That form is:

```
.FIG n
```

where n is the number of lines of blank space desired (independent of spacing). In this case no explicit figure definition is needed.

If you do not like RNF's placement of a figure, you can override it via the .FIGHERE command. It has the same syntax as .FIG; the difference is that .FIGHERE causes the figure to be printed immediately if there is space on the page and no other figures are queued. If other figures are queued or you are too near end-of-page, .FIGHERE converts to .FIG .

Outside figures, you can access the page number by referring to the predefined variable $$PAGE. This trick will not work for figures, however, because it will get the page number current at the time the figure is defined, not the page number on which the figure is eventually printed. To circumvent this problem, there is a special .ASSIGNPN command, which has the syntax:

```
.ASSIGNPN $var
```

The variable $var must have been previously declared. When .ASSIGNPN is used outside a figure definition, it is equivalent to

1. The whole text of a figure is guaranteed to appear on one page. This restricts the maximum size of a figure to somewhat less than the page size.

2. The surrounding document text "flows" around the figure, eliminating excessive white space that would result from just doing a page eject before the figure.

Figure 1. Features of Figures

Example 12 (continued)

1. Any lasting environment changes made during the definition will affect text after the definition, not after the call.

2. References to the page number of a figure must be made with the .ASSIGNPN command.

3. When the figure definition starts, the output line number (variable $$OLNO) is set to 1, so that any line number reference will be relative to the start of the figure, not the start of the page on which it is printed.

Figure 2. Aspects of Figure Definition/Call

"$var=$$PAGE;", but when used inside a figure definition, the assignment is deferred until the figure is actually printed.

There are two other special commands for use with figures:

```
     .FIGLINES figurename $var
and .FLUSHFIGS
```

.FIGLINES will assign to $var the number of lines in the indicated figure. It can be used to determine how many blank lines are needed to center a figure on a page, for example. .FLUSHFIGS is used at the end of a section to guarantee that any figures called within that section have been printed. (Otherwise, figures from chapter 3 can migrate into chapter 4, which looks terrible.) There is an implied .FLUSHFIGS command at the end of the document.

1. You can have a maximum of 200 figures and footnotes defined at any one time. It is common to define all figures at the start of the document and then just call them where they are needed. A figure becomes undefined when it is printed, so any number of figures can be handled by judicious interspersing of definitions and calls.

2. The maximum number of "outstanding" figures is 20. An outstanding figure is one that has been called but has not yet been printed.

Figure 3. Restrictions on Figures

Example 13: Footnotes

```
.PP
To get a footnote, merely bracket the text of the footnote with
the _.FOOTNOTE and _.ENDNOTE commands.  These commands cause
everything in between to be formatted as new lines and queued
to be printed at the bottom of the page.  There is no need to
count lines--RNF does that for you.  You are responsible for
linking the text with the note.  There are several common ways,
depending mostly on the capabilities of the output device you are
using.  On devices with the capability, printing superscripts is
usually the most attractive approach.  On line printers, usually
the best you can do is to follow the referenced sentence with
the footnote number in brackets.[1]# .FOOTNOTE .LM +4 .I -4
[1]#For example, this footnote references the previous sentence.
Note that the footnote can contain formatting commands in addition
to text. .LM -4 .ENDNOTE  Notice that _.FOOTNOTE does not cause
a break or any other immediate effect on the output, so it can be
used in the middle of a paragraph with impunity.
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 13 (continued)

To get a footnote, merely bracket the text of the footnote with the .FOOTNOTE and .ENDNOTE commands. These commands cause everything in between to be formatted as new lines and queued to be printed at the bottom of the page. There is no need to count lines--RNF does that for you. You are responsible for linking the text with the note. There are several common ways, depending mostly on the capabilities of the output device you are using. On devices with the capability, printing superscripts is usually the most attractive approach. On line printers, usually the best you can do is to follow the referenced sentence with the footnote number in brackets.[1] Notice that .FOOTNOTE does not cause a break or any other immediate effect on the output, so it can be used in the middle of a paragraph with impunity.

[1] For example, this footnote references the previous sentence. Note that the footnote can contain formatting commands in addition to text.

Example 14: Cross-References

```
.FLAGHYPH
.PP
This brief example illustrates how to use _.REF and _.DEF to
implement cross-/references.  Because this is a one-/page example,
cross-/references by page number would be a little trite, so
we will use cross-/references by line number to make it
more interesting.  References by page number would
be exactly the same except that they would use _$$PAGE instead
of _$$OLNO#.
.PP .DEF EX15A 2 .X
Consider line .REF REX15 ?? .  Notice how it uses a _.DEF call
to define the symbol REX15, which is used in the previous
sentence .U before .NOU it is defined.
.PP
Line $$OLNO .X :# This, obviously, .DEF REX15 $$OLNO is the line that
is being referenced by the preceeding paragraph.
.PP
When using _.DEF and _.REF, you must keep in mind that they do produce
RNF output, even though it is invisible.  As a result, an additional
command is sometimes required to get the right appearance.  With
_.DEF, you should add a _.X if the _.DEF comes at the beginning of
a paragraph (as in paragraph .REF EX15A ? .NOX above), but not if
it comes in the middle of a paragraph.  With _.REF, you should add
a _.NOX if the reference is followed by another word, but not if it
is followed by punctuation.


********** RNF OUTPUT STARTS ON NEXT PAGE
```

Example 14 (continued)

This brief example illustrates how to use .REF and .DEF to implement cross-references. Because this is a one-page example, cross-references by page number would be a little trite, so we will use cross-references by line number to make it more interesting. References by page number would be exactly the same except that they would use $$PAGE instead of $$OLNO .

Consider line 18. Notice how it uses a .DEF call to define the symbol REX15, which is used in the previous sentence before it is defined.

Line 18: This, obviously, is the line that is being referenced by the preceeding paragraph.

When using .DEF and .REF, you must keep in mind that they do produce RNF output, even though it is invisible. As a result, an additional command is sometimes required to get the right appearance. With .DEF, you should add a .X if the .DEF comes at the beginning of a paragraph (as in paragraph 2 above), but not if it comes in the middle of a paragraph. With .REF, you should add a .NOX if the reference is followed by another word, but not if it is followed by punctuation.

Example 15: Change Bars

```
.BAR
.AP .LM 1 .RM 65
.B 3
.C Using Change Bars
.B
   Change bars are used to flag text that has been altered since the
last version of a document, and are commonly found only in technical
writing, .BB where the reader must be alerted to new features. .EB
   The _.BAR command enables change bars, and has the immediate
effect of shifting all output text right
.BB three .EB
spaces.
The _.NOBAR command disables change bars.  Usually, the _.BAR command
appears before any text in a document, so the left margin in the output
is uniform.
   The _.BB (Begin Bar) command turns on change bars.  That is,
_.BB causes each subsequent line to be flagged by a _! character at the
meft margin.  The _.EB (End Bar) command turns off change bars,
signifying the end of the changed section.
   .BB Spaces caused by the _.RIGHT command appear to the left of
the three extra columns used by the _.BAR feature.
   Blank lines which appear as a part of changed text are
also flagged by a bar. .EB
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 15 (continued)

Using Change Bars

Change bars are used to flag text that has been altered since the last version of a document, and are commonly found only in
! technical writing, where the reader must be alerted to new
! features.

The .BAR command enables change bars, and has the immediate
! effect of shifting all output text right three spaces. The .NOBAR command disables change bars. Usually, the .BAR command appears before any text in a document, so the left margin in the output is uniform.

The .BB (Begin Bar) command turns on change bars. That is, .BB causes each subsequent line to be flagged by a ! character at the left margin. The .EB (End Bar) command turns off change bars, signifying the end of the changed section.

! Spaces caused by the .RIGHT command appear to the left of the
! three extra columns used by the .BAR feature.
!
! Blank lines which appear as a part of changed text are also
! flagged by a bar.

Example 16: Include Blank Lines

```
.IBL
 .PP
 The _.IBL command includes blank lines

 as they are found in the input text.

 Note that blank lines are considered to be seperate paragraphs

 by the _.AP command, autoparagraphing .

 .NOIBL
  The _.NOIBL command deletes all blank lines

 found in the text except those found in an _.ASIS

 region.

 .ASIS

      This 'asis' paragraph includes two blank lines(following),


      but they are not deleted by the _.NOIBL command because
      they occur within an _.ASIS .

!
```

********** RNF OUTPUT STARTS ON NEXT PAGE

Example 16 (continued)

The .IBL command includes blank lines

as they are found in the input text.

Note that blank lines are considered to be seperate paragraphs

by the .AP command, autoparagraphing .

The .NOIBL command deletes all blank lines found in the text except those found in an .ASIS region.

    This 'asis' paragraph includes two blank lines(following),

    but they are not deleted by the _.NOIBL command because
    they occur within an _.ASIS .

(This page is intentionally blank.)

Control Statements, Error Messages, and Utilities

How to Run RNF

Text processing on the Software Development Lab ROS computers at Langley typically involves three steps:

Step One

The input to RNF should be prepared on an M68000 machine using the ROS system and full screen editor.

Step Two

Once you have entered your text and RNF commands onto a file, you are ready to run the text through the RNF text formatting program. RNF resides as a permanent file under the ROS volume name, RNF. It may be obtained by the following control statements

```
ASSIGN RNF
PREF RNF

   or

AP RNF
```

RNF is executed by the command

```
R MAIN
```

RNF prompts the user for instructions:

```
ENTER RNF INPUT FILE NAME= carol;sampin
ENTER RNF OUTPUT FILE NAME= ctemp;documnt
TEXT OUTPUT TO TERMINAL FOR DEBUG?  <Y> <N> y
```

where:

INPUT file - the file containing the text and RNF commands.

OUTPUT file - the file containing the final formatted document.

Text output - n = no output at terminal
- y = RNF output(final document) is routed directlyto the terminal as it is generated and is interspersed with error messages for debug. Note that this can be directed to the printer or to another file (for later editing) by using:

```
R MAIN > PRINTER;
       or
R MAIN > filename.TEXT
```

Step Three

The final step involves sending the formatted document to the printer. A line printer is available at the multi-bus site as well as a letter quality printer at the Q-bus site. See below for details on how to specify each printer.

Error Messages

Error messages produced by RNF are embedded in the document as separate lines having the form

```
*** ERROR:  (explanation)  ON INPUT LINE nnnnn
```

If "TERMINAL OUTPUT FOR DEBUG" mode is selected, the output file itself contains no error messages. It just reflects whatever RNF made of your input.

If a macro file is used, then "LINE 1" would refer to the first line of the macro file. Otherwise, it refers to the first line of the input file. Illegal commands, parameters, and expressions will always be listed in the explanation in upper case. For example, the input ".ppp" will yield the explanation that ".PPP" is unrecognized.

Errors in cross-referencing, detected by DOREFS, are also embedded in the document. They have the form

```
*** ERROR -  (explanation)
```

All errors that are diagnosed by RNF and DOREFS are also flagged with one or both of the following dayfile messages:

```
***** ERRORS IN RNF INPUT *****
    *** REFERENCE ERRORS ***
```

RNF will abort if it runs out of time or memory. These conditions are diagnosed with dayfile messages that say

```
    * TIME LIMIT
or  * RUNTIME STACK OVERFLOW
```

along with a lot of other messages that you can ignore. You will receive the output that was produced before the abort. To give RNF more memory, use the RFL parameter on the RNF control card. To give RNF more time, increase your job card T parameter (for a batch job) or use the SETTL command. RNF can process even the most complicated formats in under 3 seconds per page (.5 second is typical).

Most of the errors that are diagnosed by RNF are self-explanatory, having to do with missing names, missing numbers, illegal values, etc. There is one message, however, that can be a little subtle:

```
*** ERROR:  COMMAND ILLEGAL INSIDE 'BLOCK' OF TEXT: command
```

A "block" of text is a figure (.DEFFIG through .ENDFIG), a footnote (.FOOTNOTE through .ENDNOTE), or a keep block (.KEEP through .ENDKEEP). These blocks cannot be nested--you cannot have a footnote inside a keep block, for example. Also, any command that could cause a page eject (except .PAGE) is illegal inside a block. In total, the commands that are not allowed inside a text block are .DEFFIG, .FIG, .FIGHERE, .FLUSHFIGS, .FOOTNOTE, .KEEP, .TP, .BOT, .TOP, and .MID .

This error is most frequently caused by forgetting to end a block, such as having a .KEEP without a matching .ENDKEEP .

For error messages embedded in the document, the "line number" listed is just the value of the predefined variable $$ILNO (see page 25). This variable is incremented every time a new line is read, but you can also set its value at any time with the RNF command $$ILNO=n; . If your document is broken into chapters, you may find it convenient to reset $$ILNO at the beginning of each chapter, so that chapter 1 will start with line number 1001, chapter 2 with 2001, and so on. That way, an error on "LINE 3027" clearly points to the 27-th line of chapter 3.

Printing RNF Output Files

Your RNF ASCII output may be printed on the ASCII line printer or on the letter quality printer. Normally, the line printer should be used for rough drafts and the letter quality printer should be used only for final documents.

To route a file to the letter quality printer, the file must first be transfered from the multi-bus (multi-user) computer to the Q-bus (single- user) machine. This is done by typing the following commands on the Q-bus and then the multibus machines:

Multi-bus	Q-bus
ASSIGN <volume name>	ASSIGN HUGE
ET	ET
(sending)	(receiving)

Appendices

A: Details of Parameter Substitution

This appendix supplements the discussion of macro parameters given on page 19. That discussion covers 98% of the cases. In the remaining 2% of macro uses, there are some subtle aspects of parameter substitution that can trip you up or can make the macros much easier to use and more powerful. The purpose of this appendix is to describe those subtleties so that you can be either wary or tricky, as appropriate.

First, note that the parameters of a macro are macros themselves, so they can be used by other macros and do not have to be "passed down" from one level of macro to another. They can even be referenced outside of a call to their owner macro. The parameter macros are initially defined to be null at the same time their owner is defined. The values of the parameter macros are redefined when the owner macro is called, and the parameter macros retain their new value until the owner is called again (if ever). For example, the sequence

```
.MACRO TITLE * =
.TITLE RNF Subtleties
```

causes the parameter macro TITLE1 to be assigned the value "RNF Subtleties". TITLE1 can then be called independently of TITLE and, whenever it is called, the most recently assigned value is used, as in:

```
.C Chapter 13: .TITLE1
```

which centers "Chapter 13: RNF Subtleties". This allows a parameter to be used many times.

Second, note that the parameter macros are defined at the time the owner macro is called, not after any expansion is done. Consider the sequence:

```
.MACRO GARBAGE 2 = .GARBAGE1 or .GARBAGE2
.GARBAGE .U A .NOU B
```

In this sequence, GARBAGE1 will have the value ".U" and GARBAGE2 will have the value "A". The overall expansion will therefore be

".U or A .NOU B" and the output will be "or A B", not "A or B" as you possibly might have expected. This illustrates a case in which the direct word by word parameter substitution is not what you want. All is not lost, however--the situation can be handled with an intermediate macro, as follows:

```
.MACRO UANOU = .U A .NOU
.GARBAGE .UANOU B
```

Now GARBAGE1 will have the value ".UANOU", which in turn has the value ".U A .NOU" and the expansion is as desired. If this situation comes up frequently, you might want to define a general purpose macro to "bind" words together under a macro name. This is just what the TITLE macro shown on page 99 does. So, we can define a new macro, say "BIND", to act like TITLE, and use it as follows:

```
.MACRO BIND * =
and then
.BIND .U A .NOU
.GARBAGE .BIND1 B
```

Finally, note that it is possible to substitute "nothing" for a macro parameter. If a macro is called without enough words after it to match all the parameters, then the extra parameters are defined to be null and will expand to nothing at all when they are called. For example, the sequence:

```
.MACRO Q 5 = .Q1 and .Q2 and .Q3 and .Q4 and .Q5
.Q One two three
(the end)
```

will produce "One and two and three and and (the end)". This process of assigning null parameter values is rather subtle if macros are nested. For the purpose of parameter substitution, macro expansions are not considered to be on the same line as the outermost macro call. (This is one of the exceptions mentioned earlier.) Consider, for example:

```
.Q One two three
versus
.MACRO OUTER = .Q One two
.OUTER three
```

The first case has been discussed previously--parameters Q4 and Q5 are null and the output is "One and two and three and and". In the second case, the "three" is not a candidate for substitution into Q because it does not appear in the macro that calls Q. Thus, parameter Q3 will be null in addition to Q4 and Q5, and the output will be "One and two and and and three".

B: Standard Macros

A few of the "built-in" RNF functions are in fact implemented with standard predefined macros, including .TITLE, .ST, and .CH (chaptering). You should never need to know what the standard macros are, but they are included here in case you sometime have a question about their operation. They also provide an example showing just how tricky it is possible to be. The standard macros are not easy to follow-- they make extensive use of expressions and rely on the details of parameter substitution.

At present, the standard macros are:

```
.MACRO TITLE * =
.MACRO ST * =
.MACRO FRCPAGE = .BOT .TOP ..HEAD .MID
.MACRO .HEAD = .SAV .RESPAG .B 3 .TITLE1 ..PNO .BR .ST1 .B 2 .RES
.MACRO .PNO = $$PAGE=$$PAGE+1; .IF $$NMP .TAB $$RM .RT $$PAGE
.MACRO CH * = ..CHFIRST $$CH=$$CH+1 $$HL=0; .B 2 .C .CH1 .B 3 ..SETT
.MACRO .CHFIRST = ..CLRT ..CLRST .PAGE .FIG 12 .C CHAPTER
.MACRO .SETT = .IF $$ATITLE .TITLE .CH1
.MACRO .CLRT = .TITLE
.MACRO .CLRST = .ST
```

Notice that there are many macros in this standard set having names that start with a period. These "hidden" macros are not supposed to be called directly by the user, and the function of the period is to reduce the likelihood that a standard hidden macro will conflict with a user-defined macro. As far as RNF is concerned, they would work just as well with more conventional names.

C: Formal Expression Syntax

Generally speaking, if you just follow your intuition and use lots of parentheses you will not have any trouble using expressions. However, you may at some time need more detail about what is and what is not allowed. The following is a formal description of expression syntax that you can use at that time:

```
expr      ::=    term  {relop term}

term      ::=    item  {<+|-> item}

item      ::=    <+|-|#|null> element  {<*|/> element}

element   ::=    (expr) | var | integer

var       ::=    $id <{term}|null> {=term}

relop     ::=   .EQ. | .NE. | .LT. | .LE. | .GT. | .GE.
```

In this description, braces {} are used to enclose optional parts that may be repeated any number of times, angle brackets <> enclose parts where you have a choice, the vertical bar | means "or", and spaces serve only to increase the readability. For example, the line:

```
term      ::=    item  {<+|-> item}
```

should be read as "a term is an item, optionally followed by any number of repetitions of either a plus or a minus and another item".

D: Restrictions

For the most part, RNF does a very thorough job of checking for violations of its restrictions. However, certain errors, such as exceeding the maximum input line length, are not checked and will simply produce incorrectly formatted output. A few of the restrictions are somewhat "soft", in that they can be exceeded at times without causing trouble--the values given below are lower bounds that are always safe.

The following limitations are imposed by fixed table sizes inside RNF:

Feature	Maximum number
User-defined variables	200
Tab stops	30
Environment levels (.SAV/.RES)	20
Header levels	5
List levels	10
Characters in a name (figure, variable, macro)	10
Figures and footnotes defined at one time	200
Figures outstanding (called but not yet printed)	20
Footnotes outstanding (defined but not printed)	20
Named environments	10
Length of input line (characters)	150
Length of output line (characters)	300
Characters overstruck at a single print position	4

The following restrictions are mostly imposed by the size of a line printer page. The spacing limitations are arbitrary and are used to detect commands that are almost certainly mistakes.

Parameter	Minimum	Maximum	Default
Left margin (.LM)	1	136	1
Right margin (.RM)	1	136	72
Logical page size (.PS)	7	(none)	57
Physical page size (.LINES)	0	(none)	0
Paragraph format (.P)			
indent	column 1	column 136	left margin
spacing	0	5	0
testpage	0	(none)	3
Spacing (.SP)	1	5	1
Right shift (.RIGHT)	0	136	0

E: Command Summary

This very brief command summary is intended only for quick reference. More details are given in the text on the indicated page and in the command list starting on page 41.

-B = command not legal inside figures, footnotes, or keep blocks

Function	Implied .BR or .CR	Command	Default	-B	Page
reprocess current line or macro		.AGAIN			30
set automatic paragraphing		.AP	(off)		61
declare array		.ARRAY $name size			26
start "as is" text	 	.ASIS			57
assign page number to variable		.ASSIGNPN variable			84
chapter titles are page titles		.ATITLE	(off)		42
blank lines independent of .SP	 	.B number			11
enable change bars		.BAR	(off)		91
begin change bars		.BB			91
prepare for page footer		.BOT		-B	21
break (independent of spacing)		.BR			10
center next output line	<CR>	.C			9
start new chapter	<CR>	.CH text		-B	39
carriage return		.CR			10
center word at next tab stop		.CT			64
start control sequence		.CTLSEQ			34
define name for reference		.DEF name value			36
start figure definition	<CR>	.DEFFIG name		-B	31
fill tabbed space with periods		.DOT	(off)		64
end change bars		.EB			91
end current list	<CR>	.ELIST			39
end figure definition		.ENDFIG			31
end "keep block"		.ENDKEEP			11
end footnote		.ENDNOTE			32
finish current page	<CR>	.ENDPAGE		-B	24
insert escape character (HEX1B)		.ESC			37
turn on filling		.F	(on)		10
call a figure		.FIG type		-B	31
call a figure immediately		.FIGHERE name		-B	84
assign figure length to variable		.FIGLINES name $var		-B	85

Command Summary (continued)

-B = command not legal inside figures, footnotes, or keep blocks

Function	Implied .BR or .CR	Command	Default	-B	Page
enable _ as escape character		.FLAGESC	(on)		15
enable / as phantom hyphen		.FLAGHYPH	(off)		15
enable \ as overstrike character		.FLAGOVER	(on)		15
set # as significant blank		.FLAGSIG	(on)		15
guarantee all figures printed	<CR>	.FLUSHFIGS		-B	85
print a Roman numeral or letter		.FMT n1 n2			40
start a footnote		.FOOTNOTE		-B	32
start a new header level	<CR>	.HL n heading		-B	39
indent the next line	<CR>	.I n			9
include blank lines from inpt fl		.IBL	(true)		93
test expression for nonzero		.IF expression			29
includes another file as input		.INCLUDE filename			47
turn on justification		.J	(on)		10
keep text together	 	.KEEP		-B	11
start a new list element	<CR>	.LE			39
Set physical page size		.LINES n	(0)		13
start a new list or list level	<CR>	.LIST sp [in]	[4]		39
set the left margin		.LM value	(1)		9
define a macro		.MACRO ...			18
resume text after page header	 	.MID		-B	21
turn on page numbering		.NMP	(on)		13
set paragraph format		.P in sp tst	(+0 1 3)		12
start a new page	<CR>	.PAGE			12
sentence ends with ?, !, or .		.PERIOD	(on)		15
set page number		.PNO n			13
start new paragraph	<CR>	.PP			12
set logical page size		.PS ln col	(57 72)		12
reference by name		.REF name placeholder			36
remark, skip rest of line		.REM			40
restore stacked environment		.RES			14
restore named environment		.RESENV name			14
restore page environment		.RESPAG			14
shift document to right on page		.RIGHT n	(0)		9
set right margin		.RM value	(72)		9
right-justify at next tab stop		.RT			64

Command Summary (continued)

-B = command not legal inside figures, footnotes, or keep blocks

Function	Implied .BR or .CR	Command	Default	-B	Page
blank lines dependent on .SP	<CR>	.S number			11
save environment on stack		.SAV			14
save (set) named environment		.SAVENV name			14
save (set) page environment		.SAVPAG			14
all blanks are significant		.SIG	(off)		8
set spacing between lines		.SP number	(1)		10
set page subtitle		.ST text			13
sets default options		.STANDARD			51
discard current output line		.SUP			52
start next word at next tab stop		.T			64
set tab stops		.TAB c1 ... c30	(no tabs)		64
same as .TAB		.TABS ...			64
set page title		.TITLE text			13
prepare for page header	 	.TOP		-B	21
test page for n lines remaining		.TP n		-B	11
set underscore		.U			69
underline significant blanks		.USB	(on)		69
declare numeric variable		.VAR $name ...			26
vertical tab to line n	<CR>	.VT n			53
extend (or concat.) output word		.X			8

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-86327		

4. Title and Subtitle	5. Report Date
M68000 RNF Text Formatter User's Manual	March 1985
	6. Performing Organization Code: 505-37-03-01
7. Author(s): Ralph W. Will and Carolyn Grantham	8. Performing Organization Report No.
	10. Work Unit No.
9. Performing Organization Name and Address: NASA Langley Research Center, Hampton, VA 23665	11. Contract or Grant No.
	13. Type of Report and Period Covered: Technical Memorandum
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

RNF is a powerful, flexible text formatting program. It is designed to automate many of the tedious elements of typing, including breaking a document into pages with titles and page numbers, formatting chapter and section headings, keeping track of page numbers for use in a table of contents, "justifying" lines by inserting blanks to give an even right margin, and inserting figures and footnotes at appropriate places on the page. RNF greatly facilitates both preparing and modifying a document because it allows you to concentrate your efforts on the content of the document instead of its appearance and because it removes the necessity of retyping text that has not changed.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Text Formatter Document Processor	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	109	A06

N-305

For sale by the National Technical Information Service, Springfield, Virginia 22161